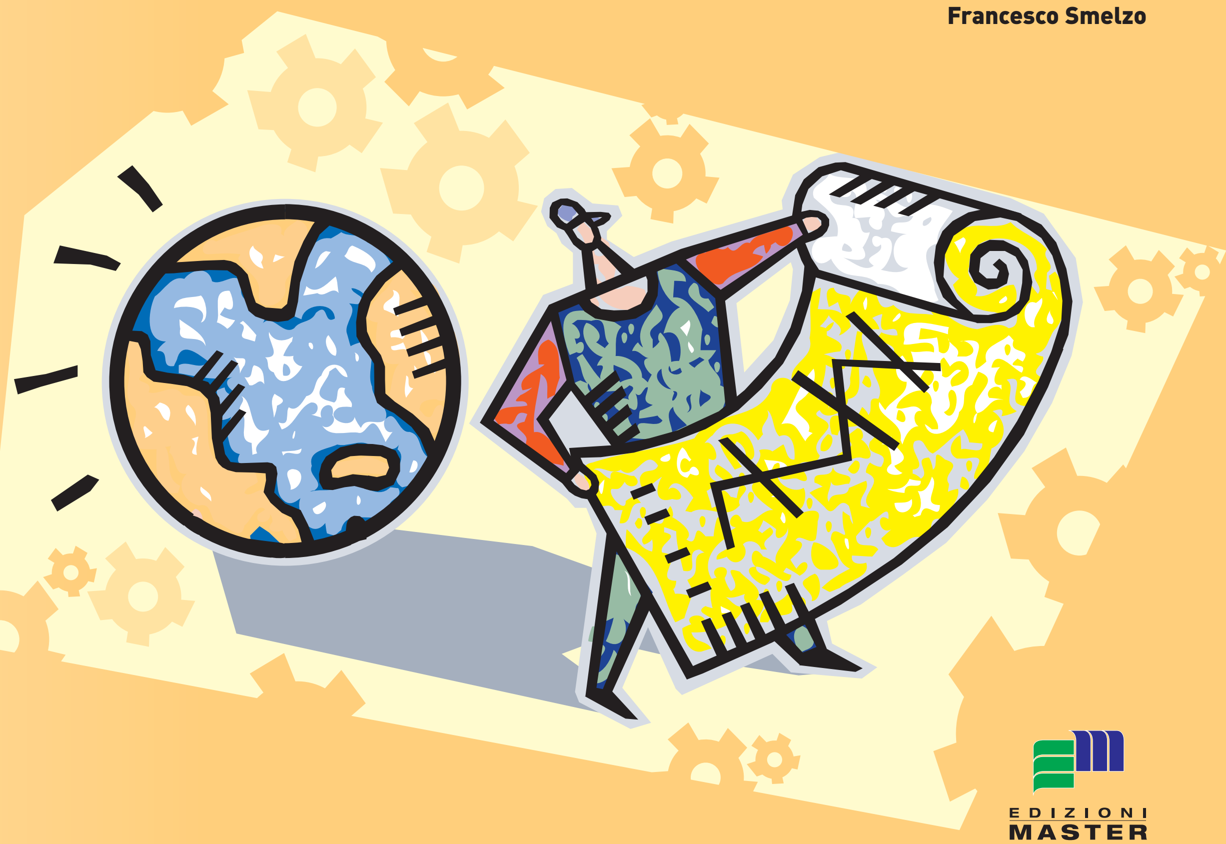
I LIBRI di

ioProgrammo

LA GUIDA PER PORTARE FACILMENTE LE TUE APPLICAZIONI NEL MONDO DEL WEB 2.0

PROGRAMMARE CON AJAX

Francesco Smelzo



EDIZIONI MASTER

i libri di

ioProgrammo

PROGRAMMARE CON AJAX

Francesco Smelzo

EDIZIONI MASTER

INDICE

Cos'è Ajax

Dom HTML

Uno sguardo a XML

Lato server

Finalmente AJAX

Ajax on the road

Considerazioni finali e riferimenti

PuntoInformatico | Libri

COS'È AJAX

Quando in programmazione si incontrano delle cose semplici e che funzionano, il riflesso incondizionato della comunità degli sviluppatori è di renderle complicate e quello delle multinazionali del software di farci anche qualche affare.

La prima cosa da sottolineare è che AJAX **NON** è un nuovo linguaggio di programmazione, ma piuttosto l'utilizzo di un insieme di tecnologie già esistenti.

Da una parte infatti c'è stata l'evoluzione dell'**HTML** che, con il supporto dei browser sempre più moderni, mette a disposizione un Modello ad Oggetti (DOM – Document Object Model) che può essere modificato, anche dopo che una pagina è stata caricata nel browser con **Javascript**, tant'è che già dalla versione 4 di internet explorer (1997) si parlava di DHTML (HTML dinamico).

Poi abbiamo avuto il "ciclone" di **XML** (1998) che è diventato lo strumento per eccellenza per il trasporto dei dati nella programmazione moderna.

Per questo insieme di tecnologie (DHTML, Javascript e XML) , nel febbraio 2005, il consulente Jesse James Garrett coniò (pare sotto la doccia) il termine di AJAX, che a noi italiani ricorda un detersivo ma in realtà è un acronimo per **Asynchronous JavaScript and XML** ovvero XML utilizzato da javascript in modo asincrono (tecnica che, come abbiamo visto, si poteva usare – e si è usata – ben prima del 2005).

1.1 FLUSSO TRADIZIONALE DELLE APPLICAZIONI WEB

Il flusso delle applicazioni Web è tradizionalmente:

1. il client richiede una certa pagina al server
2. il server elabora la richiesta e la rinvia al client

Questo significa che i "giochi" si fanno tra la richiesta (Request) e la risposta (Response), infatti tutti i vari linguaggi lato server (ASP, PHP,

JSP ecc…) non fanno altro che inserire delle istruzioni per l'elaborazione dell'output (la pagina Web) tra la Request e la Response.
In un meccanismo di questo tipo ogni aggiornamento dei dati presentati richiederà una nuova Request con il conseguente ricaricamento (e rielaborazione da parte del server) di tutta la pagina.
Pensiamo al semplice caso di una pagina web contenente una maschera per l'inserimento dei dati. Per verificare la correttezza dei dati inseriti, con la tecnica tradizionale, occorre inviarli al server dove ci sarà del codice che ne validerà la correttezza e, in caso di problemi, ricostruirà l'intera pagina indicando all'utente dove ha sbagliato.
Per mitigare il consumo di risorse in termini di banda e di CPU del server si ricorre di solito a script nella pagina web che controllino i dati in fase di invio. Ma anche questa tecnica ha le sue limitazioni, infatti è possibile controllare se un campo è vuoto o una data è sbagliata, ma non se, ad esempio, un ordine supera la quantità di merce in magazzino o se una password è già esistente nel sistema, semplicemente perché questi dati li ha il server e non il client.
L'insieme delle tecnologie AJAX invece consente di risolvere brillantemente il problema. Si carica normalmente la pagina con la maschera in questione dove sarà presente uno script che, senza ricaricare la pagina, avvia in background un dialogo con il server per verificare la correttezza dei dati che l'utente sta inserendo avvertendolo in tempo reale di un eventuale errore. Tutto questo senza bisogno di ricaricare tutto daccapo.

1.2 PERCHÉ AJAX È UNA NOVITÀ?

Come abbiamo visto gli strumenti per scrivere codice AJAX quando …"AJAX non c'era" c'erano già da tempo e possiamo anche dire (per averlo fatto) che venivano pure utilizzati.
Allora perché questa improvvisa ventata di interesse verso questa tecnologia?
Lasciando perdere il fattore "moda" (che comunque ha la sua impor-

tanza), diciamo che la "novità" di AJAX è in gran parte dovuta a due fattori:

1. Nuove applicazioni "di massa" sul web
2. Approccio allo sviluppo Web da parte dei programmatori

1.2.1 Nuove applicazioni "di massa" sul web

Le tecnologie su cui si basa AJAX anche se presenti da tempo venivano utilizzate soprattutto in ambito Intranet oppure in parti di siti web riservate a comunità ben definite, questo perché AJAX è una tecnologia fortemente basata sul browser, il quale deve avere:

1. supporto di javascript
2. supporto al DOM HTML
3. strumenti per il parsing XML

tali caratteristiche, qualche anno fa, non erano per niente scontate nei browser in circolazione, soprattutto in piattaforme non Windows. Il rischio di usare AJAX in un sito pubblico era quindi quello di tagliare fuori una fetta significativa di utenti. Per questo si preferiva utilizzare tale tecnica solamente quando si poteva "controllare" e indirizzare all'uso di un certo browser la comunità degli utenti.
Successivamente, soprattutto grazie a browser come Firefox, si è potuto disporre di una base di browser installati (Internet Explorer e Firefox da soli coprono la stragrande maggioranza dei browser utilizzati) predisposta verso AJAX.
Si è quindi incominciato ad usare tale tecnologia anche da parte in siti web di riferimento come, ad esempio, Google.
Proprio a Google si deve infatti una delle prime applicazioni AJAX "pubbliche", ovvero Google Suggest (http://www.google.com/webhp?complete=1&hl=en).
In questa applicazione, come vediamo in figura 1, man mano che l'utente digita la parola da cercare nella textbox viene mostrata una lista aggiornata delle combinazioni di parole maggiormente utilizzate dagli utenti.



Figura 1.1: Ricerca con Google Suggest

Questa ed altre applicazioni del genere, hanno portato una gran massa di sviluppatori in contatto con le possibilità offerte da AJAX ed hanno contribuito in gran parte ad accrescere l’interesse verso di esso.

1.2.2 Approccio allo sviluppo Web da parte dei programmatori

Tuttavia nel far vedere AJAX come un fenomeno nuovo ha inciso anche un altro, non secondario fattore.
Com’è che si arriva a diventare sviluppatori Web? Beh, solitamente ci sono due “canali” :

1. il web-designer che dopo aver giocato in tutti i modi possibili con l’HTML si accorge che questo, da solo, non basta ed inizia ad esplorare un linguaggio di scripting e poi magari una tecnologia “lato server” come ASP,PHP,JSP ecc…
2. il programmatore di applicazioni desktop tradizionali (in VB, DELPHI, C++ ecc…) che pian piano deve, per esigenze di

mercato, spostarsi sul web.
Anche se il punto d'arrivo di queste due "categorie" di sviluppatori è lo stesso le condizioni di partenza, e quindi il background culturale, è ben diverso.
Il programmatore del secondo tipo non avrà problemi a gestire connessioni a database, algoritmi sofisticati ecc… ma resta praticamente "in panne" quando ha a che fare con il mondo dei TAG, dei CSS e degli altri standard del mondo Web. Basti pensare quant'è ancor oggi difficile far digerire ad un programmatore tradizionale la sintassi, tutto sommato banale, di XML.
Naturalmente anche il programmatore del primo tipo avrà le sue lacune, ma sicuramente con la sintassi HTML e simili si trova molto più "a casa sua".
C'è da dire che delle due tipologie di programmatori Web la seconda è in larghissima parte prevalente, e questo lo sta a dimostrare chiaramente l'evoluzione delle varie tecnologie lato server, ASP.NET, JSP e da ultimo pare anche PHP.
Le tecnologie di sviluppo Web di ultima generazione, infatti, tendono a riprodurre il più fedelmente possibile l'ambiente di sviluppo tradizionale. Ecco che si parla quindi di Controlli , di Eventi, di Finestre ecc…
Tutto questo per "nascondere" il più possibile allo sviluppatore le particolarità, e le peculiarità dell'ambiente in cui si trova ad operare per farlo assomigliare sempre di più ad un normale programma desktop. Ecco quindi che ci si "inventa" una miriade di POST nascosti (che fanno crescere a dismisura le dimensioni della pagina da scaricare sul browser) per conservare lo stato dei controlli e gestire gli eventi.
Il risultato è che il programmatore perde progressivamente il contatto con l'ambiente in cui opera effettivamente e, pur scrivendo per il Web, magari incontra grandissima difficoltà ad inserire un semplice script di validazione.
Allora è chiaro che un programmatore Web di questo tipo che si tro-

va davanti una tecnologia tutta incentrata sul Client come AJAX si trova come un pesce fuor d'acqua e comincia a cercare spasmodicamente controlli, tools di generazione automatica o di conversione e quant'altro che la fantasia umana può produrre.

1.3 LO SCOPO DEL LIBRO

Fatta questa premessa, parliamo di noi.
Questo libro nasce con il preciso scopo di portare, passo passo, il programmatore a riprendere confidenza con l'ambiente Web.
Il libro non vuole essere una somma di nozioni (anche se un po' di teoria la dovremo per forza illustrare), ma piuttosto vuol partire da semplici casi pratici per arrivare fino ai primi esempi di AJAX.
Incontreremo quindi diverse tecnologie : il DOM HTML e CSS, XML e XPATH visti sempre dal punto di vista di javascript.
Per lavorare in javascript con XML, gestito da molti browser ma con modalità radicalmente differenti, vedremo anche alcune librerie di funzioni che semplificano il lavoro.
Poi, prima di addentrarci nell'AJAX vero è proprio, ci soffermeremo sul versante server con degli esempi in PHP, C# e VB.NET (i concetti esposti si possono comunque applicare a qualsiasi linguaggio).
Non pretendiamo di essere esaustivi su ogni tecnologia trattata (altrimenti dovremmo avere a disposizione uno spazio ben maggiore di un istant book), ma di guidare il lettore attraverso gli esempi,di stimolarne l'approfondimento e dare degli spunti nell'ottica di risolvere casi concreti.
Sempre per esigenze di spazio, e anche per non complicare troppo le cose, nel libro non trattiamo un'altra delle tecnologie XML che può essere utile con AJAX e cioè XSLT (eXtensible StyLesheet Tranformations) ovvero la trasformazione diretta di XML (magari in HTML) nella convinzione che sia più utile dare piuttosto un

quadro di riferimenti generali.

1.4 CONOSCENZE RICHIESTE

La lettura del libro richiede una conoscenza di base della sintassi javascript acquisibile anche attraverso la lettura di uno dei tanti tutorial reperibili in rete (sempre ottima la sezione Web Development della library di Microsoft - http://msdn.microsoft.com/library), preferibile anche una certa frequentazione di XML (può essere utile il manuale su XML di IoProgrammo uscito di recente) e qualche nozione di programmazione lato server.

DOM HTML

HTML è sostanzialmente un insieme di regole di formattazione del testo basato su speciali marcatori, o TAG, che definiscono gli elementi da visualizzare.
Ad esempio, il codice per visualizzare una tabella con una riga e due celle sarà:

```
<table> <!--inizia tabella -->
    <tr> <!--inizia riga-->
        <td> <!--prima Cella-->
            Nome <!--testo -->
        </td><!-- fine prima Cella-->
        <td><!-- seconda Cella -->
            Cognome <!--testo -->
        </td><!-- fine seconda Cella-->
    </tr> <!--fine riga-->
</table><!--fine tabella-->
```

Ogni TAG ha poi una serie di proprietà che definiscono lo stile di visualizzazione: la larghezza di un elemento, il colore di sfondo ecc…
Queste proprietà sono scritte come attributi del TAG, per cui volendo dare alla nostra tabella una larghezza di 400 pixel, alla prima cella il colore di sfondo grigio e alla seconda giallo potremo scrivere:

```
<table width="400">
    <tr>
        <td bgcolor="#EEEEEE">
            Nome
        </td>
        <td bgcolor="#FFFF66">
            Cognome
```

```
        </td>
    </tr>
</table>
```

L'attributo è espresso all'interno del marcatore di apertura nella forma nome="valore" , il valore è espresso tra virgolette semplici ' o doppie " (anche se in HTML, a differenza che in XML, le virgolette non sono obbligatorie per i valori senza spazi). I valori relativi ai colori sono espressi in formato esadecimale, la stringa del colore rappresenta i tre numeri, ognuno da 0 (ovvero 00) a 255 (ovvero FF), che compongono lo spettro RGB (Rosso , Verde, Blu) dei colori primari del monitor, quindi il bianco che è 255-255-255 in esadecimale sarà #FFFFFF.
Ogni TAG, detto anche Elemento, contiene decine di proprietà. L'elemento TABLE, ad esempio, contiene (tra le altre) queste proprietà:

Attributo	**Descrizione**
ALIGN	Allineamento tabella
BACKGROUND	Immagine di sfondo
BGCOLOR	Colore di sfondo
BORDER	Spessore bordo
BORDERCOLOR	Colore bordo
CELLPADDING	Spazio interno alle celle
CELLSPACING	Spazio tra una cella e l'altra
HEIGHT	Altezza
WIDTH	Larghezza

Tutti gli elementi hanno poi degli attributi comuni, tra cui:

Attributo	**Descrizione**
ID	Identificativo univoco dell'elemento
Class	Classe CSS
Style	Stile CSS inline

L'affollamento di attributi necessari per la formattazione di una pagina web ha portato poi alla nascita dei fogli di stile che non sono altro che raccolte di proprietà raggruppate per:

- Nome del TAG
- Nome del Gruppo di regole (Classe)
- ID dell'elemento

Le regole CSS sono espresse nella forma:

```
nome:valore;nome2:valore2;
```

E possono essere scritte direttamente nel TAG (modalità inline) usando l'attributo style:

```
<table style="width:400px;background-color:#FFFFFF">
oppure inserite nel tag <style> nella pagina stessa:
<style>
TABLE {
    width:400px;
    background-color:#FFFFFF;
}
.myTable {
    width:400px;
    background-color:#FFFFFF;
}
#table1 {
    width:400px;
    background-color:#FFFFFF;
}
</style>
```

qui abbiamo inserito le tre modalità di referenziare un gruppo di regole:

- la prima si applicherà a tutti gli elementi che hanno il nome

TABLE

- la seconda si applicherà a tutti gli elementi che hanno l'attributo class uguale al nome assegnato (senza punto) come: <table class="myTable">
- la terza si applicherà all'elemento che ha l'id uguale a quello indicato (senza cancelletto) come: <table id="table1">

Le regole CSS, oltre all'interno del TAG (nell'attributo style) e nel TAG <style>, possono essere definite anche in file separati da quello della pagina Web e poi collegate alla pagina "madre" con il TAG <link> :

```
<link type="text/css" rel="stylesheet" href="stili.css">
```

2.1 ED IL DOM?

Fin qui un breve ripasso (degno di un Bignami) dell'HTML, ma il DOM cos'è?
In pratica con è altro che la modellizzazione dei vari TAG come gerarchia di oggetti.
Come primo oggetto abbiamo Window che rappresenta una finestra del browser (vedi schema in figura 2), ogni Window avrà un Document che rappresenta l'HTML del documento corrente,

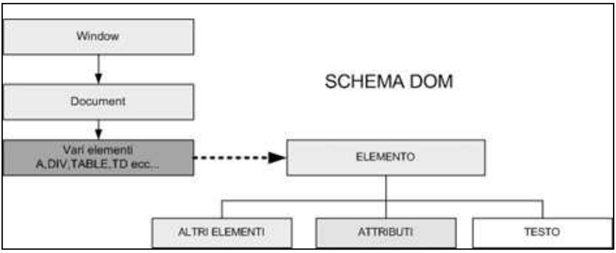


Figura 2: schema sintetico DOM

il quale avrà una serie di oggetti Element nidificati fino a rappresentare l'albero HTML.
Naturalmente le cose sono un po' più complesse di quanto enunciato, tuttavia ai nostri scopi ci basta sapere che:

1. tutti gli elementi sono visti come oggetti
2. tutti gli attributi degli elementi sono le proprietà degli oggetti
3. tutti gli oggetti espongono anche degli eventi

Occorre ora soffermarci sul concetto di Nodo. Il nodo rappresenta l'entità principale del DOM, relativamente all'HTML sono nodi:

- il Document stesso
- gli Elementi o TAG
- gli Attributi degli elementi
- il Testo interno agli elementi

Un tipo di nodo si riconosce dall'altro per la proprietà nodeType che definisce, a seconda del valore, le varie tipologie :
1 = Elemento
2 = Attributo
3 = Testo

Il concetto di nodo è importante perché attraverso un nodo di possono trovare i nodi figlio e il nodo padre.
I nodi che possono contenere sotto-elementi (quindi nodi Elemento o il Document) hanno infatti la proprietà childNodes che rappresenta l'insieme dei sotto-elementi e del testo e, i nodi elemento, hanno la proprietà attributes che rappresenta l'insieme degli attributi.
Parimenti un nodo ha la proprietà parentNode che consente di risalire all'elemento padre.
Altre proprietà comuni a tutti i nodi sono:
nodeName – ovvero il nome del

Per capire meglio rifacciamoci all'esempio di codice HTML visto in precedenza:

```
<table width="400">
    <tr>
        <td bgcolor="#EEEEEE">
            Nome
        </td>
        <td bgcolor="#FFFF66">
            Cognome
        </td>
    </tr>
</table>
```

ovvero, tradotto in DOM :

Nome nodo	Tipo	Valore	Parent	ChildNodes
TABLE	Element	-	-	TR
WIDTH	Attribute	400	TABLE	-
TR	Element	-	TABLE	TD,TD
TD	Element	-	TR	#TEXT
BGCOLOR	Attribute	#EEEEEE	TD	-
#TEXT	Testo	Nome	TD	-
TD	Element	-	TR	#TEXT
BGCOLOR	Attribute	#FFFF66	TD	-
#TEXT	Testo	Cognome	TD	-

Tutto questo si può esprimere in javascript con iterazioni del tipo:

```
var body = document.body;
for(var i=0;i<body.childNodes.length;i++){
    var node = body.childNodes[i];
```

```
        if(node.nodeType==1) ;//element
        else; //text
}
```

2.2 TROVARE GLI ELEMENTI

Per andare a trovare l'istanza di un elemento HTML dichiarato nella pagina abbiamo visto che è possibile implementare una serie di cicli For ed esaminare le proprietà nodeType e nodeValue ciò però è decisamente poco pratico perché spesso gli elementi si trovano a vari livelli di nidificazione.
Per questo il DOM mette a disposizione una serie di metodi molto utili:

- getElementById
- getElementsByName
- getElementsByTagName

getElementById

getElementById è il metodo dell'oggetto document che consente di referenziare un elemento presente nella pagina dato il suo attributo ID.

Prendiamo sempre lo stesso esempio di HTML , inserendo l'attributo ID in una cella:

```
<table id="myTable" width="400">
    <tr>
        <td id="primaCella" bgcolor="#EEEEEE">
            Nome
        </td>
        <td bgcolor="#FFFF66">
            Cognome
        </td>
```

```
    </tr>
</table>
```

A questo punto sarà possibile riferirsi, nel codice javascript, a questo elemento come:

```
var cell1 = document.getElementById("primaCella");
```

L'id deve naturalmente essere univoco per ogni elemento, nel caso in cui vi siano più elementi con lo stesso id verrebbe restituito solamente il primo.

getElementsByName

getElementsByName è anch'esso un metodo di document che però consente di recuperare non un singolo elemento ma un'insieme di elementi (Array) dato il valore del loro attributo name.
Dato che l'attributo name è proprio solo degli elementi di input utente (form, input, select e textarea) la tecnica è applicabile solo a questi.
Partendo, ad esempio, da questo codice HTML :

```
<input type="text" name="box">
<input type="text" name="box">
<input type="text" name="box2">
```

è possibile recuperare i primi due elementi input con:

```
var boxes = document.getElementsByName("box");
```

getElementsByTagName

getElementsByTagName, al contrario degli altri due metodi che abbiamo visto, non è proprio soltanto dell'oggetto document, ma di tutti gli elementi. Esso consente di recuperare tutti i sottoe-

lementi con un TAG di un determinato nome.
Sempre rifacendosi all'esempio della tabella vista in precedenza, possiamo recuperare il riferimento alla table con getElementById e quindi tutte le celle con getElementsByTagName:

```
var oTable = document.getElementById("myTable");
var cells = document.getElementsByTagName("TD");
```

il nome dell'elemento può essere scritto indifferentemente in maiuscolo o in minuscolo.

2.3 TROVARE E IMPOSTARE GLI ATTRIBUTI

Abbiamo visto come sia facile trovare e referenziare attraverso javascript gli elementi della pagina. Altrettanto facile è trovare gli attributi che popolano l'elemento individuato con il metodo getAttribute che è comune a tutti gli elementi.
Prendiamo ad esempio questo frammento HTML :

```
<div id="div1" title="prova">testo</div>
```

Volendo recuperare da javascript il valore dell'attributo title dell'elemento DIV sarà sufficiente:

```
var oDiv = document.getElementById("div1");
var attValue = oDiv.getAttribute("title");
```

Altrettanto semplice è impostare la proprietà di un attributo, con il metodo setAttribute :

```
var oDiv = document.getElementById("div1");
oDiv.setAttribute("title","nuovo valore");
```

setAttribute prevede due parametri, il primo è il nome dell'attributo da impostare o creare ed il secondo è il valore che deve assumere.

Le proprietà

getAttribute e setAttribute sono i metodi standard che offre il DOM per recuperare e impostare gli attributi, con questi metodi possiamo recuperare e impostare attributi anche non presenti nel linguaggio HTML.
Se ad esempio avessimo un TAG HTML come:

```
<div id="div1" categoria="prova">testo</div>
```

potremmo egualmente recuperare e impostare l'attributo categoria con getAttribute e setAttribute anche se esso non è previsto dal linguaggio HTML. Questa è una risorsa importante per lo sviluppatore perché consente di estendere i TAG secondo le proprie esigenze.
Tuttavia quando abbiamo un riferimento ad un elemento HTML attraverso il DOM l'oggetto risultante espone tutti gli attributi standard del linguaggio anche come proprietà.
Quindi per recuperare o impostare l'attributo title (che è un attributo proprio di DIV) avremmo potuto tranquillamente usare:

```
var oDiv = document.getElementById("div1");
var attValue = oDiv.title; // lettura
oDiv.title = "nuovo valore"; // scrittura
```

Praticamente ogni attributo che può essere inserito negli elementi HTML si ritrova come proprietà negli oggetti referenziati con DOM con lo stesso nome, fa eccezione l'attributo **class** utilizzato per riferirsi a un gruppo di regole CSS, in questo caso infatti la proprietà di chiama className.
Quindi per impostare la classe CSS di un elemento dinamica-

mente dovremmo scrivere :

```
var oDiv = document.getElementById("div1");
oDiv.className="miaClasse";
```

Diverso è il discorso per gli stili in linea che, come abbiamo visto, in HTML si esprimono così:

```
<div id="div1" style="background:#FFFFFF;text-
                                  align:left;color:#0099EE">
```

nel DOM per ogni elemento esiste la proprietà style che però è un oggetto a sé stante che ha a sua volta una serie di proprietà per quante sono le regole definibili con i CSS.
Quindi potremo impostare gli stili inline di un elemento con :
var oDiv = document.getElementById("div1");

```
oDiv.style.background = "#FFFFFF";
oDiv.style.textAlign = "left";
oDiv.style.color = "#0099EE";
```

2.4 CREARE E RIMUOVERE I NODI

Finora abbiamo visto come sia possibile, grazie a DOM, manipolare gli elementi che esistono nella pagina HTML, ma le potenzialità di DOM non si esauriscono certo qui!
Il modello dispone infatti di una serie di metodi per creare da zero nuovi nodi e aggiungerli ad altri già esistenti o, viceversa, rimuovere quelli già esistenti.

Creazione di nodi

I metodi per la creazione di nuovi nodi appartengono all'oggetto document e sono:

- createElement
- createTextNode

c’è anche createAttribute, ma è reso pressoché inutile dal più comodo setAttribute che abbiamo già visto.
Con createElement è possibile creare da zero un nuovo oggetto e impostarne che proprietà. Ecco come creare un nuovo elemento DIV utilizzando il DOM con javascript:

```
var oDiv = document.createElement("div");
oDiv.id = "div1";
oDiv.style.background = "#FFFFFF";
oDiv.style.textAlign = "left";
oDiv.style.color = "#0099EE";
```

Il parametro del metodo createElement è appunto il nome dell’elemento.
Nello stesso modo è possibile anche creare dinamicamente un nodo di testo (cioè del testo) :

```
var txt = document.createTextNode("Questo è un testo");
```

dove il parametro sarà invece la stringa valore del nodo.
Possiamo così creare nuovi elementi o nodi di testo ma se ci fermassimo qui questi oggetti verrebbero solamente instanziati in memoria e non visualizzati nel browser. Per far sì che questo succeda occorre collocarli, cioè inserirli all’interno di un elemento.
Per questa operazione tutti gli elementi dispongono di alcuni metodi:

- appendChild – che consente di inserire un nodo sotto ad un altro
- insertBefore – che consente di inserire un nodo prima di un altro

● replaceNode – che consente di sostituire un nodo con un altro

Qui ci limiteremo a prendere in considerazione il primo metodo e vediamo come aggiungere un nuovo elemento a questo DIV esistente :

```
<div id="container"></div>
```

In javascript creeremo quindi un nuovo DIV e lo inseriremo sotto quello esistente con :

```
var oDiv = document.createElement("div");
oDiv.id = "div1";
oDiv.style.background = "#FFFFFF";
oDiv.style.textAlign = "left";
oDiv.style.color = "#0099EE";
var divEsistente = document.getElementById("container");
divEsistente.appendChild(oDiv); //appende l'elemento oDiv
```

Il risultato visibile nel browser sarà quindi come se noi avessimo scritto in HTML :

```
<div id="container">
<div id="div1" style="background:#FFFFFF;text-
                                   align:left;color:#0099EE"></div>
</div>
```

innerHTML

La creazione di nodi con createElement non è però l’unico modo di aggiungere dinamicamente dei contenuti agli elementi già esistenti, il DOM dota infatti gli elementi della proprietà innerHTML che può essere usata per aggiungere frammenti di co-

dice HTML in forma di stringa a elementi già esistenti.
Il codice che abbiamo visto in precedenza, per aggiungere un DIV sotto ad un altro, con innerHTML, si sarebbe potuto esprimere con:

```
var divEsistente = document.getElementById("container");
divEsistente.innerHTML='<div id="div1"
                                        style="background:#FFFFFF;' +
                  'text-align:left;color:#0099EE"></div>';
```

si risparmiano un bel po' di righe di codice ma si rende lo script meno lineare.
innerHTML ha però anche altri risvolti pratici, ad esempio può essere usato per inserire del testo direttamente senza usare il metodo document.createTextNode() .
Con createTextNode per aggiungere del testo avremmo dovuto scrivere:

```
var oDiv = document.createElement("div");
oDiv.id = "div1";
oDiv.style.background = "#FFFFFF";
oDiv.style.textAlign = "left";
oDiv.style.color = "#0099EE";
var txt = document.createTextNode("Questo è un testo");
oDiv.appendChild(txt);
var divEsistente = document.getElementById("container");
divEsistente.appendChild(oDiv); //appende l'elemento oDiv
```

Mentre usando innerHTML possiamo scrivere direttamente:

```
var oDiv = document.createElement("div");
oDiv.id = "div1";
oDiv.style.background = "#FFFFFF";
```

```
oDiv.style.textAlign = "left";
oDiv.style.color = "#0099EE";
oDiv.innerHTML="Questo è un testo";
var divEsistente = document.getElementById("container");
divEsistente.appendChild(oDiv); //appende l'elemento oDiv
```

La proprietà innerHTML viene poi utilizzata anche per rimuovere tutto il contenuto di un elemento (sotto-elementi e nodi di testo) in modo rapido.
Se abbiamo del codice HTML come questo :

```
<div id="container">
    <div id="div1" style="background:#FFFFFF;text-
                    align:left;color:#0099EE">testo</div>
    <div id="div2" style="background:#FFFFFF;text-
                    align:left;color:#0099EE">Altro testo</div>
</div>
```

possiamo rimuovere tutto il contenuto del DIV principale semplicemente con :

```
var divEsistente = document.getElementById("container");
divEsistente.innerHTML= "";
```

ed il risultato sarà :

```
<div id="container"></div>
```

Rimuovere i nodi

Al di là del piccolo trucco che abbiamo visto per la rimozione del contenuto di un elemento utilizzando la proprietà innerHTML il DOM offre anche un modo più “orientato agli oggetti”.
Ogni elemento dispone infatti dei metodi:

- removeChild – per rimuovere un nodo figlio
- removeNode – per rimuovere il nodo stesso

Quindi per rimuovere il DIV con id “div2” da:

```
<div id="container">
    <div id="div1" style="background:#FFFFFF;text-
                    align:left;color:#0099EE">testo</div>
    <div id="div2" style="background:#FFFFFF;text-
                    align:left;color:#0099EE">Altro testo</div>
</div>
```

potremo usare :

```
var divContainer = document.getElementById("container");
var div2 = document.getElementById("div2");
divContainer.removeChild(div2);
```

oppure:

```
var div2 = document.getElementById("div2");
div2.removeNode();
```

2.5 MANIPOLAZIONE DI TABELLE

Le tabelle in HTML si definiscono con i seguenti elementi:

- table – definisce una tabella
- tr – definisce una riga in una tabella
- td – definisce una cella in una riga (al posto di td è possibile usare th per le celle della riga di intestazione).

Quindi il codice HTML di una tabella potrebbe essere :

```
<table> <!--inizia tabella -->
    <tr> <!--inizia riga-->
```

```
        <td> <!--prima Cella-->
            Nome <!--testo -->
        </td><!-- fine prima Cella-->
        <td><!-- seconda Cella -->
            Cognome <!--testo -->
        </td><!-- fine seconda Cella-->
    </tr> <!--fine riga-->
</table><!--fine tabella-->
```

Una cosa che gli sviluppatori a volte ignorano o dimenticano è poi che esistono degli elementi che raggruppano le righe in base alla loro collocazione:

- THead – Testa della tabella
- TBody – Corpo della tabella
- TFoot – Piede della tabella

Praticamente è un po' come avviene in Word con l'intestazione e il piè di pagina.
Se questi elementi non vengono definiti in HTML il parser del Browser li aggiunge automaticamente, inserendo tutte le righe nel TBody.
L'esempio precedente sarebbe letto dal browser come :

```
<table> <!--inizia tabella -->
<thead></thead>
<tbody>
    <tr> <!--inizia riga-->
        <td> <!--prima Cella-->
            Nome <!--testo -->
        </td><!-- fine prima Cella-->
        <td><!-- seconda Cella -->
            Cognome <!--testo -->
```

```
            </td><!-- fine seconda Cella-->
      </tr> <!--fine riga-->
</tbody>
<tfoot></tfoot>
</table><!--fine tabella-->
```

Questo dettaglio è importante perché quando, attraverso il DOM, abbiamo il riferimento a una riga (elemento tr):

```
var oTr = document.getElementById("riga1");
```

ci si aspetterebbe che il nodo Parent fosse l'elemento table:

```
var oTr = oTr.parentNode;
```

e invece no, perché nel DOM il parent di tr è sempre tbody, quindi per trovare il riferimento alla tabella avremmo dovuto scrivere:

```
var oTr = oTr.parentNode.parentNode;
```

Essendo composta di elementi come gli altri, una tabella può essere gestita con il DOM.
Ad esempio possiamo crearla con:

```
var oTable = document.createElement("table");
var oTr = document.createElement("tr");
var oTd = document.createElement("td");
oTd.innerHTML= "Nome";
oTr.appendChild (oTd);
var oTd = document.createElement("td");
oTd.innerHTML= "Cognome";
oTr.appendChild (oTd);
```

```
oTable.appendChild (oTr);
```

Table Object Model

Gli elementi che compongono la tabella, probabilmente per il largo uso che se ne fa, oltre al DOM (che resta sempre utilizzabile) dispongono anche di un altro modello ad oggetti: il Table Object Model.
Cioè, in pratica, hanno dei metodi e proprietà proprie. Vediamo quali.

Metodi per la manipolazione di tabelle

createTHead	Crea un elemento tHead nella tabella.
deleteTHead	Cancella un elemento tHead dalla tabella.
createTFoot	Crea un elemento tFoot element nella tabella.
deleteTFoot	Cancella un elemento tFoot dalla tabella.
insertRow	Crea una nuova riga nella tabella e la aggiunge all'insieme rows.
deleteRow	Cancella la riga dalla tabella e la rimuove dall'insieme rows.
insertCell	Crea una nuova cella in una riga e la aggiunge all'insieme cells.
deleteCell	Cancella la cella da una riga e la rimuove dall'insieme cells.

insertRow e insertCell, hanno anche un parametro che definisce la posizione in cui inserire l'oggetto, il valore -1 indica che deve essere inserito in coda. Per cui, ad esempio, il codice:

```
var oTr = oTable.insertRow(-1);
```

Inserisce una riga in coda alle altre della tabella.

Insiemi

tBodies	Insieme degli oggetti tBody nella tabella.
Rows	Insieme degli oggetti tr (riga) nella tabella.
Cells	Insieme degli oggetti td (cella) nella riga di una tabella o nell'intera tabella.

Righe e celle hanno poi proprietà di indice che indicano la loro posizione.

Indici

sourceIndex	Posizione dell'elemento nella gerarchia di una tabella.
rowIndex	Indice della riga nel contesto della tabella a cui appartiene.
sectionRowIndex	Indice della riga nel contesto all'interno della sezione THead,TBody o TFoot.
cellIndex	Indice della cella nel contesto della riga a cui appartiene.

Con questi strumenti lavorare sulle tabelle diventa più semplice. Ecco come si presenterebbe il nostro codice usando il Table Object Model:

```
var oTable = document.createElement("table");
var oTr = oTable.insertRow(-1);
var oTd = oTr.insertCell(-1);
oTd.innerHTML= "Nome";
var oTd = oTr.insertCell(-1);
oTd.innerHTML= "Cognome";
```

Notate come i metodi insert di righe e celle provvedano direttamente all'append dell'oggetto all'insieme relativo.

2.6 GLI EVENTI

Arrivati a questo punto abbiamo imparato a referenziare gli elementi HTML con oggetti con il DOM e a manipolarli attraverso attributi e proprietà.

Ci manca però ancora una cosa importante : collegare i nostri scripts che utilizzano il DOM ad azioni dell'utente o a eventi del browser. Gli eventi non sono altro che segnali (o messaggi) che l'applicazione manda al verificarsi di certe condizioni, se a questi

segnali sono associate delle funzioni script queste saranno eseguite.
In HTML gli eventi sono espressi come normali attributi dei TAG. Un bottone, ad esempio, può dichiarare una funzione associata all'evento onclick semplicemente con:

```
<button id="btn1" onclick="cliccami()">Clicca qui!</button>
```

nella pagina ci sarà quindi uno script all'interno del quale verrà definita la funzione cliccami():

```
<script language="javascript" type="text/javascript">
function cliccami(){
     alert("Mi hai cliccato!");
}
</script>
```

2.7 ASSOCIAZIONE DI EVENTI CON IL DOM

L'associazione evento/funzione può essere effettuata anche con il DOM in tre modi diversi:

```
var oButton = document.getElementById("btn1");
//primo modo
oButton.onclick = cliccami;
//secondo modo
oButton.onclick = function (){
     alert("Mi hai cliccato!");
}
//terzo modo
oButton.onclick = new Function('alert("Mi hai cliccato!")');
```

tutti tre i modi assegnano una funzione all'evento tuttavia:

- il primo modo assegna un riferimento ad una funzione definita da qualche altra parte (si riconosce che è un riferimento perché sono assenti le parentesi)
- il secondo modo assegna direttamente una funzione costruita sul posto
- il terzo modo utilizza il costruttore new per un nuovo oggetto Function che ha il corpo passato come parametro

Il secondo ed il terzo modo sono utili in particolare con il DOM. Il secondo è decisamente il più usato tuttavia presenta un inconveniente particolare : all'interno della funzione costruita le variabili esterne non contano più.
Cercheremo di spiegarci meglio con un esempio.
Abbiamo un ciclo con cui si crea una serie di bottoni, cliccando su ogni bottone deve apparire un messaggio con un numero progressivo diverso, quindi implementiamo il seguente codice:

```
for(var i=0;i<4;i++){
    var oButton = document.createElement("button");
    oButton.value= "bottone " + i;
    oButton.onclick = function (){
        alert("Hai premuto il bottone " + i);
    }
}
```

bene, questo codice non funzionerà come desiderato perché all'interno della funzione la variabile i non ha alcun senso poiché la funzione verrà eseguita nel contesto dell'elemento non in quello globale.
In questi casi si deve quindi ricorrere al costruttore di funzione che permettere di comporre il corpo della stessa come stringa:

```
for(var i=0;i<4;i++){
```

```
    var oButton = document.createElement("button");
    oButton.value= "bottone " + i;
    oButton.onclick = new Function (
        'alert("Hai premuto il bottone "' + i + ')'
        );
}
```

Associazione multipla di eventi

Quella che abbiamo visto prima è la modalità di associazione singola, evento/funzione.

I browser però supportano anche l'associazione multipla, cioè più funzioni associate allo stesso evento.

Purtroppo però lo fanno in maniera radicalmente differente.

Internet Explorer

Internet Explorer segue una strada proprietaria attraverso il metodo attachEvent disponibile per tutti gli oggetti.

Vediamo come assegnare più eventi in Internet Explorer:

```
var oButton = document.createElement("button");
oButton.attachEvent("onclick",cliccami);
oButton.attachEvent("onclick",function (){
    alert("secondo messaggio")
});
```

come possiamo vedere il primo argomento del metodo è il nome letterale dell'evento, mentre il secondo è una funzione indicata come riferimento o scritta direttamente.

Firefox e gli altri

I browser più "ligi" agli standard realizzano invece l'associazione multipla ad un evento con il metodo addEventListener, anch'esso proprio di tutti gli oggetti DOM.

L'uso è simile a quello di attachEvent :

```
var oButton = document.createElement("button");
```

```
oButton.addEventListener("click",cliccami,false);
oButton.addEventListener("click",function (){
    alert("secondo messaggio")
},false);
```

Come possiamo notare il nome dell'evento è espresso senza il prefisso "on" e c'è un terzo parametro booleano; il terzo parametro se impostato a true indica che l'evento viene gestito direttamente dall'oggetto e non viene notificato ai livelli superiori.
Se, ad esempio, abbiamo un DIV nidificato in un altro e tutti e due avessero associata una funzione all'evento onclick l'associazione con addEventListener effettuata sull'elemento più interno, se dichiarata con il terzo parametro a true, significherebbe che il click sull'elemento non darebbe luogo all'azione prevista dall'evento assegnato all'elemento superiore.

Mettere d'accordo tutti

In questo come in altri casi dello scripting Client-side le differenze tra i vari browser trasformano la vita del programmatore in un inferno, per fortuna c'è una libreria di funzioni chiamata IEEmu (***http://webfx.eae.net/dhtml/ieemu***) che permette di utilizzare la sintassi di Internet Explorer anche con gli altri browser.

onload

Tutti gli oggetti del DOM hanno associati una serie di eventi, quello che però per noi ha un'importanza cruciale è l'evento onload associato all'oggetto window.
Per capirne meglio l'importanza analizziamo la seguente pagina HTML :

```
<html>
<head>
    <title>Esempio</title>
```

```
    <script language="javascript" type="text/javascript">
        var div1 = document.getElementById("div1");
        alert(div1.innerHTML);
    </script>
</head>
<body>
<div id="div1">Testo</div>
</body>
</html>
```

come ormai avrete capito lo script dovrebbe trovare l'elemento con id “div1” e mostrarne il contenuto in un messaggio.
Questo script invece non funzionerà. Non funzionerà perché, trovandosi a livello globale, verrà eseguito prima ancora che il documento HTML sia stato completamente caricato ed analizzato dal browser, in questa fase il DOM non è disponibile (proprio perché il documento non è stato ancora analizzato).
Ed è qui che entra in gioco l'evento onload; onload infatti è un evento che si verifica quando il documento è completamente caricato e analizzato e quindi DOM è disponibile.
Il nostro codice quindi, per funzionare, avrebbe dovuto essere scritto così:

```
<html>
<head>
    <title>Esempio</title>
    <script language="javascript" type="text/javascript">
        window.onload = function () {
            var div1 = document.getElementById("div1");
            alert(div1.innerHTML);
        }
    </script>
</head>
```

```
<body>
<div id="div1">Testo</div>
</body>
</html>
```

Quindi : quando le azioni che coinvolgono il DOM non sono collegate ad altro evento (come il click su un bottone ecc…) devono essere eseguite nel contesto dell'evento onload di window.

Altri eventi

Ogni oggetto DOM ha associati una serie di eventi; qualcuno specifico per la funzione che deve svolgere quel determinato elemento, altri comuni un po' a tutti gli elementi.
Tra gli eventi più comuni troviamo:

- onkeydown – quando l'utente preme un tasto
- onkeypress – quando l'utente preme un tasto alfanumerico
- onkeyup – quando l'utente rilascia un tasto
- oncontextmenu – quando l'utente fa click con il tasto destro
- onclick – quando l'utente fa click sull'oggetto con il tasto sinistro del mouse
- ondblclick – quando l'utente fa doppio click sull'oggetto con il tasto sinistro del mouse
- onmousedown – quando l'utente fa click sull'oggetto con qualsiasi tasto del mouse
- onmouseenter – quando l'utente si sposta con il mouse nell'area dell'oggetto
- onmouseleave – quando l'utente lascia con il mouse nell'area dell'oggetto
- onmousemove – quando l'utente si muove con il mouse nell'area dell'oggetto
- onmouseout – quando l'utente lascia con il mouse i limiti dell'area dell'oggetto
- onmouseover – quando l'utente entra con il mouse nei limi-

ti dell'area dell'oggetto
- onmouseup – quando l'utente rilascia qualsiasi tasto del mouse, in precedenza premuto, nell'area dell'oggetto

2.8 METTIAMO TUTTO INSIEME

Con quello che abbiamo visto fin qui non avremo imparato tutto quello che c'è da sapere sul DOM HTML ma dovremmo avere le "carte in regola" per un primo semplice esempio su cose che ci torneranno utili con AJAX.
Problema
Creare una pagina HTML con un BUTTON e un DIV vuoto.
Sull'evento click associato al BUTTON si deve compiere questa azione:
- Se il DIV è vuoto creare una TABLE e inserirla sotto di esso
- Se il DIV è nascosto visualizzarlo, viceversa se è visibile nasconderlo

Soluzione
Passo 1 – creare la pagina HTML
In un editor di testo creiamo la seguente pagina HTML:

```
<html>
  <head>
       <title>Esempio</title>
            <script language="javascript" type="text/javascript">
             //codice javascript
            </script>
  </head>
  <body>
     <button id="btn1" onclick="cliccami()">Clicca qui per vedere la
     tabella!</button>
     <div id="container" style="display:none"></div>
  </body>
```

```
</html>
```

Nel browser il risultato sarà quello indicato in figura A

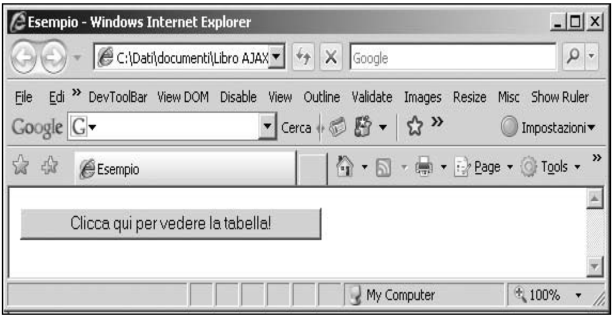


Figura A: a pagina HTML di esempio

Notare come abbiamo definito degli id per ogni elemento, lo stile CSS inline con la proprietà display su none (nascosto) per il DIV e l'associazione all'evento onclick per BUTTON.

Passo 2 – codice DOM per la creazione della tabella

All'interno dell'elemento <script> dichiariamo un oggetto globale currentTable, inizialmente a null, e scriviamo una funzione createTable che crea dinamicamente la tabella , con dati presi da un array, l'appende al DIV e l'assegna alla variabile globale currentTable:

```
var currentTable = null;
 function createTable(){
      //definizione dati tabella
      //prima riga
      var headers = ["nome","cognome","indirizzo"];
      //dati strutturati come Array di Array
      var data = [
```

```
        ["Mario", "Rossi" , "via Dante 2 Milano"],
        ["Alfredo", "Bianchi" , "via Verga 5 Firenze"],
        ["Giovanni", "Verdi" , "via Foscolo 10 Roma"],
        ["Francesco", "Gialli" , "via Pascoli 7 Napoli"]
    ];
    var oTable = document.createElement("TABLE");
    //proprietà tabella
    oTable.width = "400";
    oTable.cellSpacing = "0";
    oTable.cellPadding = "2";
    //prima riga
    var oTr = oTable.insertRow(-1);
    for(var i=0;i<headers.length;i++){
        var oTd = oTr.insertCell(-1);
        oTd.align="center";
        oTd.background = "#EEEEEE";
        oTd.style.font = "bold 12px arial";
        oTd.style.borderBottom = "1px solid #666666";
        oTd.innerHTML= headers[i];
    }
    //righe dati
    for(var i=0;i<data.length;i++){
        var rowData = data [i]; //array riga dati
        var oTr = oTable.insertRow(-1);
        for(var n=0;n<rowData.length;n++){
            var oTd = oTr.insertCell(-1);
            oTd.style.font = "12px verdana";
            oTd.style.borderBottom = "1px solid #999999";
            oTd.innerHTML= rowData[n];
        }
    }
    currentTable = oTable;
    var container = document.getElementById("container");
```

```
    container.appendChild(currentTable);
}
```

Scriviamo poi la funzione cliccami, già associata all'evento onclick di BUTTON, che si occupa di richiamare createTable se currentTable è null e di gestire la visibilità del DIV:

```
function cliccami(){
    //instanzia i riferimenti
    var container = document.getElementById("container");
    var btn1 = document.getElementById("btn1");
    //controlla currentTable ed eventualmente la crea
    if(currentTable==null) createTable();
    //imposta visibilità del DIV
    if (container.style.display=="none") {
        container.style.display="";
        btn1.value = "Clicca qui per nascondere la tabella!"
        }
    else {
        container.style.display="none";
        btn1.value = "Clicca qui per vedere la tabella!"
        }
}
Passo 3 – controllare il risultato
Alla fine il codice completo della nostra pagina sarà :
<html>
  <head>
     <title>Esempio</title>
        <script language="javascript" type="text/javascript">
         var currentTable = null;
         function createTable(){
             //definizione dati tabella
             //prima riga
```

```
var headers = ["nome","cognome","indirizzo"];
//dati strutturati come Array di Array
var data = [
      ["Mario", "Rossi" , "via Dante 2 Milano"],
      ["Alfredo", "Bianchi" , "via Verga 5 Firenze"],
      ["Giovanni", "Verdi" , "via Foscolo 10 Roma"],
      ["Francesco", "Gialli" , "via Pascoli 7 Napoli"]
];
var oTable = document.createElement("TABLE");
//proprietà tabella
oTable.width = "400";
oTable.cellSpacing = "0";
oTable.cellPadding = "2";
//prima riga
var oTr = oTable.insertRow(-1);
for(var i=0;i<headers.length;i++){
      var oTd = oTr.insertCell(-1);
      oTd.align="center";
      oTd.background = "#EEEEEE";
      oTd.style.font = "bold 12px arial";
      oTd.style.borderBottom = "1px solid #666666";
      oTd.innerHTML= headers[i];
}
//righe dati
for(var i=0;i<data.length;i++){
      var rowData = data [i]; //array riga dati
      var oTr = oTable.insertRow(-1);
      for(var n=0;n<rowData.length;n++){
            var oTd = oTr.insertCell(-1);
            oTd.style.font = "12px verdana";
            oTd.style.borderBottom = "1px solid #999999";
            oTd.innerHTML= rowData[n];
      }
```

```
            }
            currentTable = oTable;
            var container = document.getElementById("container");
            container.appendChild(currentTable);
        }

        function cliccami(){
            //instanzia i riferimenti
            var container = document.getElementById("container");
            var btn1 = document.getElementById("btn1");
            //controlla currentTable ed eventualmente la crea
            if(currentTable==null) createTable();
            //imposta visibilità del DIV
            if (container.style.display=="none") {
                container.style.display="";
                btn1.value = "Clicca qui per nascondere la tabella!"
                }
            else {
                container.style.display="none";
                btn1.value = "Clicca qui per vedere la tabella!"
                }
        }
        </script>
  </head>
  <body>
    <button id="btn1" onclick="cliccami()">Clicca qui per vedere la
    tabella!</button>
    <div id="container" style="display:none"></div>
  </body>
</html>
```

Se eseguiamo la pagina nel browser, cliccando sul bottone potremo vedere la tabella che appare (o, se visibile, scompare) co-

me in figura B



Figura b: il nostro codice in azione

UNO SGUARDO A XML

Dopo il breve “ripasso” di javascript e del DOM HTML non possiamo trascurare il secondo ingrediente di AJAX: l’XML.
L’XML è una modalità di formattazione del testo a marcatori molto simile all’HTML. A differenza di quest’ultimo però:

- è libero nel contenuto, in quanto possiamo usare elementi e attributi con nomi arbitrari
- è più rigido nella sintassi

Mentre infatti per un documento XML ci possiamo “inventare” i nomi di elementi e attributi come in :

```
<biblioteca>
    <reparto nome="narrativa">
        <libro>Guerra e pace</libro>
        <libro>Odissea</libro>
    </reparto>
</biblioteca>
```

Per la scrittura di un documento Well Formed dobbiamo seguire poche ma importanti regole sintattiche, un documento non Well Formed diventa infatti inutilizzabile dai programmi che dovranno utilizzarlo:

Regola numero 1 – elemento radice

Il documento deve avere uno ed un solo elemento radice (root element) ad esempio:

```
<biblioteca>
    <libro>Guerra e pace</libro>
<libro>Odissea</libro>
</biblioteca>
```

Abbiamo uno e un solo elemento (biblioteca) come radice.
Un documento invece del tipo:

```
<libro>Guerra e pace</libro>
<libro>Odissea</libro>
```

Sarebbe errato perché ha due elementi radice.

Regola numero 2 – tag finali obbligatori

Quando un elemento contiene testo o altri elementi deve essere chiuso con il corrispondente tag finale. Ad esempio:

```
<libro>Guerra e pace</libro>
```

Dove l'elemento <libro> deve trovare corrispondenza nel tag di chiusura </libro> allo stesso livello.
Senza il tag di chiusura verrebbe generato un errore dai programmi che tentino di leggerlo.

Regola numero 3 – elementi vuoti

Se un elemento è vuoto, nel senso che non contiene testo o altri elementi, la chiusura può avvenire anche con la sintassi abbreviata: <**libro/**>.
Che sarebbe equivalente a <**libro**><**/libro**>.

Regola numero 4 – attributi tra virgolette

Gli attributi devono essere racchiusi tra virgolette doppie (") o semplici ('). Entrambe le virgolette utilizzate devono essere dello stesso tipo (doppie o semplici).
Ad esempio, questo metodo per definire gli attributi è corretto:

```
<libro autore="Verga">I Malavoglia</libro>
```

Questo invece è errato

```
<libro autore=Verga>I Malavoglia</libro>
```

Regola numero 5 – elementi non sovrapposti

Gli elementi non possono essere chiusi prima di chiudere i sotto-elementi in essi contenuti.
Questo è un esempio di sintassi non valida:

```
<biblioteca>
    <narrativa>
        <libro autore="Verga">
            I Malavoglia
    </narrativa>
        </libro>
</biblioteca>
```

Perché l'elemento **narrativa** viene chiuso prima dell'elemento **libro** che inizia al suo interno.
La sintassi corretta sarebbe stata invece:

```
<biblioteca>
    <narrativa>
        <libro autore="Verga">
            I Malavoglia
        </libro>
    </narrativa>
</biblioteca>
```

Regola numero 6 – gli elementi sono case-sensitive
I TAG di apertura e di chiusura devono avere il nome scritto con lettere dello stesso tipo (maiuscole/minuscole), ad esempio l'elemento: <libro></libro> è corretto, mentre non lo è:<LIBRO></libro>.

3.1 IL DOM XML

Ebbene sì, anche XML come HTML ha un suo Document Object Model (anzi quello di HTML deriva per buona parte da XML) che è implemen-

tato dai programmi che analizzano l'XML (parser).

I nodi
Anche qui ritroviamo il concetto di Nodo che ha tre proprietà fondamentali:

- nodeType – il tipo di nodo
- nodeValue – il valore del nodo
- nodeName – il nome del nodo

L'oggetto Node è la base per tutti gli altri oggetti che ne derivano per estensione (attributi, elementi ecc…). Esso dispone quindi di proprietà e di metodi comuni anche agli oggetti derivati:

Proprietà

attributes	Un oggetto NamedNodeMap contenente tutti gli
	attributi di un nodo
childNodes	Un oggetto NodeList contenente tutti i nodi figlio
firstChild	Il primo nodo figlio
lastChild	L'ultimo nodo figlio
namespaceURI	L' URI del namespace di un nodo
nextSibling	Il nodo immediatamente seguente a un nodo allo stesso livello
nodeName	Il nome di un nodo
nodeType	Il tipo (costante numerica) di un nodo
nodeValue	Il valore del nodo
ownerDocument	L'oggetto Document di un nodo
parentNode	Il nodo che contiene il nodo
prefix	Il prefisso del namespace di un nodo
previousSibling	Il nodo immediatamente precedente a un nodo allo stesso livello

Metodi

appendChild(newnode)	Aggiunge un nuovo nodo figlio a un nodo e lo restituisce
cloneNode(boolean) hasChildNodes()	Crea una copia esatta di un nodo. Se il parametro è true riproduce anche i nodi figli
insertBefore(newnode,refnode)	Restituisce true o false a seconda del il nodo ha nodi figli

removeChild(nodename)	Rimuove il nodo figlio specificato e lo restituisce
replaceChild(newnode,oldnode)	Sostituisce il vecchio nodo (oldnode) con il nuovo (newnode) e restituisce il vecchio nodo

Ovviamente tali proprietà e metodi avranno un senso diverso a seconda del tipo di nodo a cui si applicano. Ad esempio la proprietà attributes di un nodo di tipo Attr (attributo) restituirà **null** ecc…

Dall'oggetto Node derivano, come abbiamo detto, gli altri oggetti che rappresentano particolari tipi di nodi, tra cui document, element e attribute.

Oggetto Document (derivato da Node)

Document rappresenta il documento XML stesso.

Proprietà

doctype	Il DTD o lo Schema del documento
documentElement	L'oggetto Element che rappresenta l'elemento radice del documento

Metodi

createAttribute("name")	Crea un nuovo nodo attributo
createCDATASection("text")	Crea un nuovo nodo CDATA
createComment("text")	Crea un nuovo nodo commento
createDocumentFragment()	Crea un oggetto vuoto documentFragment
createElement("name")	Crea un nuovo nodo elemento
createEntityReference("name")	Crea un nuovo nodo entityReference
createProcessingInstruction (target,text)	Crea un nuovo nodo processingInstruction
createTextNode("text")	Crea un nuovo nodo text
getElementById("id")	Restituisce il nodo corrispondente a un id
getElementsByTagName ("name")	Restituisce un oggetto NodeList di tutti gli oggetti Nodo che hanno il nome uguale a quello specificato.

Oggetto Element (derivato da Node)

Element rappresenta un nodo di tipo elemento, un elemento XML ad esempio:

```
<autore>Omero</autore>
```

Se un elemento ha del testo contenuto questo viene visto come nodo figlio di tipo testo (non come **nodeValue**).

Proprietà

tagName	Nome dell'elemento (lo stesso valore di nodeName)

Metodi

getAttribute(name)	Restituisce il valore dell'attributo specificato
getAttributeNode(name)	Restituisce l'attributo specificato come oggetto Attribute
getElementsByTagName(name)	Restituisce un oggetto NodeList di tutti gli oggetti Nodo che hanno il nome uguale a quello specificato.
hasAttribute()	Restituisce true o false a seconda se l'elemento ha attributi
normalize()	Unisce tutti i nodi di Testo sottostanti in un unico nodo di Testo
removeAttribute(name)	Rimuove il valore dell'attributo specificato
removeAttributeNode(name)	Rimuove l'attributo specificato
setAttribute(name,value)	Imposta il valore dell'attributo specificato
setAttributeNode(name)	Inserisce un nuovo attributo

Oggetto Attribute (derivato da Node)

Rappresenta l'attributo di un elemento.
Ad esempio in:

```
<prezzo valore="30"/>
```

L'oggetto **attribute** sarà corrispondente a valore.

Proprietà

name value	nome dell’attributo (lo stesso valore di nodeName)
	Restituisce o imposta il valore dell’attributo

3.2 DOM XML E JAVASCRIPT

Abbiamo detto che XML DOM (come HTML DOM) non è che un’interfaccia di programmazione per accedere, in lettura e scrittura ai dati contenuti in un documento XML.
Questa operazione viene compiuta da un parser che non è altro che un programma che analizza un documento XML e restituisce un oggetto DOM che può essere manipolato attraverso proprietà e metodi.
Esistono dei parser praticamente per tutti i linguaggi di programmazione, noi però, per utilizzare AJAX, dovremo accedere al DOM XML attraverso javascript.
Javascript, com’è logico essendo un linguaggio di script, non ha un parser XML integrato nell’engine, si appoggia invece a quelli che gli mette a disposizione il Browser.
E qui vengono le “dolenti note”! Come al solito i vari browser mettono a disposizione canali differenti per l’accesso a XML. Internet Explorer lo fa attraverso la chiamata ad un oggetto ActiveX mentre Firefox ed altri attraverso dei metodi integrati.

Accesso al DOM con javascript

Per capire meglio urge un esempio.
Prendiamo i dati che abbiamo utilizzato nell’esempio del capitolo precedente e trasformiamoli in un documento XML che salveremo in una cartella come data.xml:

```
<root>
    <persona nome="Mario" cognome="Rossi" indirizzo="via Dante 2
```

```
                                                          Milano"/>
   <persona nome="Alfredo" cognome="Bianchi" indirizzo="via Verga
                                                        5 Firenze"/>
   <persona nome="Giovanni" cognome="Verdi" indirizzo="via
                                                   Foscolo 10 Roma"/>
   <persona nome="Francesco" cognome="Gialli" indirizzo="via
                                                  Pascoli 7 Napoli"/>
</root>
```

Internet Explorer

Vediamo come Internet Explorer consente di accedere ai dati come oggetto DOM:

```
   function apriDomIE(docPath){
         var objDOM = new ActiveXObject("Msxml2.DOMDocument");
         objDOM.async=false
         objDOM.load(docPath);
         return objDOM;
   }
```

In pratica si instanzia l'engine nativo MSXML, installato nei sistemi Windows, e si utilizza il metodo load per caricare il documento.
I dati sono stati caricati in modo sincrono, il flusso cioè aspetta fino a che il caricamento non viene completato, ma IE offre anche la possibilità di caricamenti asincroni:

```
var xmlDoc=null;
function apriDomIEAsync(docPath){
  xmlDoc =  new ActiveXObject("Msxml2.DOMDocument");
  xmlDoc.onreadystatechange = readystatechange;
  xmlDoc.load(docPath);
}
function readystatechange(){
```

```
  var state = xmlDoc.readyState;
  if (state == 4) {
    var nome = xmlDoc.documentElement.firstChild.getAttribute("nome");
      alert(nome);
  }
}
apriDomIEAsync("data.xml");
```

In pratica l'oggetto viene dichiarato a livello globale (variabile xml-Doc) e nella funzione che lo istanzia è associata una funzione di callback all'evento onload quindi viene fatto partire il caricamento, quando esso termina viene eseguita la funzione di callback con sull'oggetto che nel frattempo sarà stato valorizzato.

Firefox

In Firefox invece il caricamento del file XML avviene in sempre modo asincrono, quindi avremo:

```
var xmlDoc=null;
function apriDomFF(docPath){
        var state = 0;
        xmlDoc = document.implementation.createDocument("", "", null)
        xmlDoc.onload = _loaded;
        xmlDoc.load(docPath);
}
function _loaded(){
      var nome =
                        xmlDoc.documentElement.firstChild.getAttribute("nome");
      alert(nome);
}
apriDomFF("data.xml");
```

Asincrono o sincrono?

I puristi affermano che l'implementazione di Firefox è la migliore il caricamento asincrono consente di non bloccare l'interfaccia utente mentre si stanno caricando i dati da un file remoto.
In parte ciò è vero, ma a volte nella vita reale le cose sono un po' diverse.
Pensate ad un caso niente affatto ipotetico: una funzione deve caricare un piccolo file XML contenente le preferenze dell'utente, leggerle e individuare la path di un file di dati più grande e caricarlo.
Con solo il metodo asincrono avremmo due caricamenti in successione, con due callback nidificati e con un notevole aumento della complessità del codice (per niente giustificato, visto che il primo file era solo di pochi Kb).
Quindi, siamo d'accordo che il metodo asincrono è generalmente il migliore, ma perché non lasciarci la possibilità di scegliere? A volte l'interfaccia utente deve rimanere bloccata fino a che non abbiamo acquisito i dati!

Caricamento di XML da una stringa

Abbiamo visto come caricare un file XML, ma a volte potrebbe essere necessario creare un oggetto DOM da una stringa XML in memoria.
Anche qui (verrebbe da dire : "naturalmente") i metodi adottati dai vari browser sono diversi.

Internet Explorer

In Internet Explorer il caricamento da stringa si fa con il metodo loadXml dell'oggetto DOMDocument, ovvero:

```
function loadXmlIE(){
      var xmlDoc = new ActiveXObject("Msxml2.DOMDocument");
      xmlDoc.async = false;
      s ='<root><persona nome="Mario" cognome="Rossi"
                              indirizzo="via Dante 2 Milano"/>';
```

```
    s+='    <persona nome="Alfredo" cognome="Bianchi"
indirizzo="..."/>';
    s+='    <persona nome="Giovanni" cognome="Verdi"
                                        indirizzo="via Foscolo 10 Roma"/>';
    s+='    <persona nome="Francesco" cognome="Gialli"
                                              indirizzo="via Pascoli ..."/>';
    s+='</root>';
xmlDoc.loadXml(s);
    var nome =
            xmlDoc.documentElement.firstChild.getAttribute("nome");
    alert(nome);
}
```

Firefox

In Firefox invece si invoca un oggetto DOMParser, ovvero:

```
function loadXmlFF(){
    s ='<root><persona nome="Mario" cognome="Rossi"
                                        indirizzo="via Dante 2 Milano"/>';
    s+='    <persona nome="Alfredo" cognome="Bianchi"
                                                      indirizzo="..."/>';
    s+='    <persona nome="Giovanni" cognome="Verdi"
                                        indirizzo="via Foscolo 10 Roma"/>';
    s+='    <persona nome="Francesco" cognome="Gialli"
                                              indirizzo="via Pascoli ..."/>';
    s+='</root>';
    var objDOMParser = new DOMParser();
    var xmlDoc = objDOMParser.parseFromString(s, "text/xml");
    var nome =
            xmlDoc.documentElement.firstChild.getAttribute("nome");
    alert(nome);
}
```

Manipolazione dei dati

In qualsiasi modo abbiate ottenuto il vostro oggetto DOM potete finalmente star sicuri che proprietà e metodi standard saranno disponibili: infatti malgrado le differenze nel caricamento dei dati l'oggetto che ne risulterà avrà certamente quelle caratteristiche di cui abbiamo parlato prima.

Ci saranno quindi i metodi per leggere e impostare gli attributi (getAttribute e setAttribute), gli insiemi di elementi e nodi di testo childNodes, i metodi per creare nuovi elementi (createElement) e aggiungerli a nodi esistenti (appendChild) e così via…

Tuttavia, con il solo DOM XML, una cosa risulta particolarmente scomoda: trovare il riferimento ad un nodo o ad un insieme di nodi.

Pensate dover trovare tutti gli elementi <libro> appartenti al reparto "Classici" in un file come questo:

```
<?xml version="1.0"?>
<biblioteca>
 <reparto nome="Classici">
  <libro>
   <autore>Omero</autore>
   <titolo>Odissea</titolo>
  </libro>
  <libro>
   <autore>Omero</autore>
   <titolo>Iliade</titolo>
  </libro>
 </reparto>
 <reparto nome="Fantasy">
  <libro>
   <autore>J.R.R. Tolkien</autore>
   <titolo>Il signore degli Anelli</titolo>
   <editore>Mondadori</editore>
  </libro>
```

```
 </reparto>
</biblioteca>
in javascript dovreste scrivere qualcosa tipo :
var doc = loadXml ("biblioteca.xml");
var reparti = doc.getElementsByTagName("reparto")
for(i=0;i<reparti.length;i++){
   var reparto = reparti[i]; //nodo <reparto>
   if( reparto.getAttribute("nome")=="Classici") {
      var libri = reparto.getElementsByTagName("libro")
      for(j=0;j<libri.length;j++){
         var libro = libri[j]; //nodo <libro>
         //...
      }
   }
}
```

cioè implementare una serie di cicli nidificati con regole interne di confronto per selezionare le informazioni.
No. Decisamente serve qualcosa di più pratico …
Per vostra (e nostra) fortuna esiste XPath che è un linguaggio simile a quello che per SQL sono le query di selezione.

3.3 XPATH

XPath è dotato di una sintassi basata sui percorsi (Path appunto) che consente di selezionare e filtrare i nodi in base alla loro posizione, ai loro attributi ecc…
La sua filosofia sta nel tradurre tutto in path, nel file che abbiamo visto in precedenza, ad esempio, i nodi <libro> hanno tutti la path: biblioteca/reparto/libro e attraverso questa path sarà possibile ottenere un insieme di nodi (nodeList).

relazioni tra i nodi≤

XPath prende in considerazione innanzitutto le relazioni tra i nodi

che possono essere :

- Parent - ogni Element (tranne quello radice) , Attribute o TextNode ha un genitore
- Children - i nodi di tipo Element possono avere uno o più nodi figli
- Siblings - i nodi di tipo Element hanno una relazione Sibling con gli elementi di pari livello.
- Ancestor - sono Ancestors (ascendenti) di un elemento tutti gli elementi che lo precedono, fino alla radice (che è il primo elemento)
- Descendants - sono Descendants (discendenti) di un elemento tutti gli elementi che racchiude

Selezione dei nodi

XPath è espressione di un percorso (come le path del sistema operativo) in un documento XML.

Il nodo è selezionato seguendo il suo percorso. Ecco alcune delle "path expressions" più utilizzate:

Espressione	Descrizione
nome	Seleziona tutti gli elementi figli del nodo con il nome corrispondente a quello indicato
/	Seleziona dal nodo radice
//	Seleziona i nodi nel documento dal nodo corrente che corrisponde alla selezione indipendente dalla posizione in cui siamo
.	Seleziona il nodo corrente
..	Seleziona nodo Parent del nodo corrente
@nome	Rappresenta un attributo con il nome corrispondente a quello indicato

Esempi :

path expression	Risultato
/biblioteca	Tutti i nodi <biblioteca> a partire dalla radice
/biblioteca/reparto	Seleziona tutti i nodi < reparto> sotto a <biblioteca> a partire dalla radice
//autore	Seleziona tutti i nodi <autore> indipendentemente da dove si trovino
//titolo/@lingua	Seleziona l'attributo lingua di tutti i nodi <titolo>

Predicati

I predicati sono usati per trovare un nodo specifico o un nodo che contiene (o ha un attributo che contiene) un dato valore.
I predicati devono essere contenuti in parentesi quadre [].

Esempi:

path expression	Risultato
/biblioteca/reparto [1]	Seleziona il primo nodo <reparto> sotto a <biblioteca>
/biblioteca/reparto [last()]	Seleziona l'ultimo nodo <reparto> sotto a <biblioteca>
//titolo[@lingua='en']	Seleziona i nodi <titolo> che hanno l'attributo "lingua" con valore "en"
//libro[titolo/@lingua='en'] /autore	Seleziona i nodi <autore> sotto <libro> che ha l'attributo "lingua" con valore "en" nell'elemento <titolo>

Selezionare nodi non conosciuti

Dei metacaratteri (wildcards) possono essere usati al posto del nome del nodo

Espressione	Descrizione
*	Qualsiasi elemento
@*	Qualsiasi attributo
node()	Qualsiasi nodo di ogni tipo

Esempi:

path expression	Risultato
/biblioteca/*	Seleziona tutti gli elementi sotto a <biblioteca> indipendentemente dal nome
/biblioteca/reparto/@*	Seleziona tutti gli attributi di <reparto> sotto a <biblioteca> indipendentemente dal nome

Selezioni multiple

L'utilizzo dell'operatore "|" in XPath consente selezioni in più percorsi.

Esempi:

path expression	Risultato
//libro/titolo | //libro/autore	Seleziona sia <titolo> che <autore> sotto a <libro>

3.4 XPATH E JAVASCRIPT

A noi, per AJAX, XPath interessa ovviamente dal punto di vista client-side e quindi con javascript. Come per DOM anche per XPath javascript si affida al motore offerto dal browser e quindi anche qui abbiamo modi di accesso diversi (la sintassi delle espressioni naturalmente non cambia) tra Internet Explorer e Firefox.

Internet Explorer

Internet Explorer usa il motore XPath integrato nella stessa libreria di DOM XML, in quest'ultimo ogni nodo di tipo document o di tipo element ha disposizione due metodi :

- selectNodes – per selezionare, data un'espressione XPath, una nodeList.
- selectSingleNode – per selezionare, data un'espressione XPath,

un singolo nodo.
Per selezionare un nodo <persona> il cui attributo nome sia uguale a “Mario” dal file data.xml che raccoglieva la lista di indirizzi che abbiamo visto in precedenza in Internet Explorer è sufficiente:

```
function selezionaIE(docPath){
    var objDOM = new ActiveXObject("Msxml2.DOMDocument");
    objDOM.load(docPath);
    var node = objDOM.selectSingleNode("//persona[@nome='Mario']");
    return node;
}
```

per selezionare invece la lista di nodi <persona> :

```
function selezionaListaIE(docPath){
    var objDOM = new ActiveXObject("Msxml2.DOMDocument");
    objDOM.load(docPath);
    var nodes = objDOM.selectNodes("//persona");
    for(var i=0;i<nodes.length;i++){
        var node = nodes[i];
        //fai qualcosa con il nodo
    }
}
```

Firefox
In Firefox le cose sono ben diverse, XPath richiede un Namespace Resolver e una serie di altri parametri, poi occorrerà implementare la funzione come callback. Questo è il codice per la selezione di una nodeList :

```
var xmlDoc=null;

function apriDomFF(docPath){
```

```
        xmlDoc = document.implementation.createDocument("", "", null)
        xmlDoc.onload = _loaded;
        xmlDoc.load(docPath);
}

function _loaded(){
       var NSResolver =
                          xmlDoc.createNSResolver(xmlDoc.documentElement);
       var oResult = xmlDoc.evaluate("//persona",
                   xmlDoc,
                   NSResolver,
                   XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);
       for(i=0; i< oResult.snapshotLength; i++) {
              var node = oResult.snapshotItem(i);
              //fai qualcosa con il nodo
       }
}
```

3.5 XMLHTTPREQUEST

L'ultimo (ma forse il più importante) tra gli strumenti per AJAX che andiamo ad analizzare è XMLHttpRequest ovvero quell'oggetto che consente di inviare una richiesta (request) HTTP al server e ricevere una risposta (response).

A prima vista XMLHttpRequest sembrerebbe un doppione dei vari metodi per caricare un documento XML che abbiamo già visto parlando del DOM.

Si potrebbe pensare infatti che per ricevere un documento XML sia in fondo sufficiente invocare il metodo load dell'implementazione DOM del browser.

In realtà le cose stanno in modo ben diverso, il load del DOM infatti :

1. effettua solo richieste http di tipo GET e non POST

2. non consente (o lo consente in maniera troppo limitata) l'invio di parametri al server

In sostanza effettuare il load di un documento XML dal server (anche se quest'ultimo fosse generato dinamicamente) consente solamente un flusso unidirezionale server/client, il client non invia informazioni al server ma si limita a riceverle.
In qualche caso questo può essere sufficiente, ma in applicazioni AJAX interattive tipicamente è il client a condizionare (in base ai parametri che invia nella request) le informazioni che poi riceve.
Inoltre il metodo POST non soffre delle limitazioni alla lunghezza imposte invece al metodo GET e quindi risulta più utile per inviare una grande quantità di informazioni.
Per supportare AJAX ci serve qualcosa di più robusto e affidabile del semplice load di XML e XMLHttpRequest fa appunto al caso nostro.
Anche l'oggetto XMLHttpRequest come DOM e XPath deriva da uno standard definito dal W3C quindi i metodi e le proprietà che espone sono gli stessi, e cioè:

Proprietà

onreadystatechange	Associa un gestore di evento da chiamare quando cambia la proprietà readyState
readyState	Codice numerico che rappresenta lo stato della richiesta.
responseText	Risposta del server in forma di stringa.
responseXML	Risposta del server in forma di oggetto DOM XML.
status	Il codice numerico di status HTTP ricevuto dal server.
statusText	Descrizione dello status HTTP ricevuto dal server.

Metodi

abort	Cancella la richiesta HTTP in corso.
getAllResponseHeaders	Recupera i valori di tutti gli headers HTTP.
getResponseHeader	Recupera il valore di un header HTTP dalla risposta.
open	Inizializza una richiesta specificando il metodo, la URL e la modalità (sincrona o asincrona) della richiesta.
send	Invia una richiesta HTTP al server (con i parametri) e riceve una risposta.
setRequestHeader	Specifica un header HTTP.

Il flusso di una richiesta sincrona con XMLHttpRequest è:

- creare un istanza dell'oggetto XMLHttpRequest
- invocare il metodo open con i parametri metodo, url e il valore false per qualificare le richiesta come sincrona
- se la richiesta è di tipo POST utilizzare setRequestHeader per impostare l'header HTTP "Content-Type" sul valore "application/x-www-form-urlencoded"
- invocare il metodo send con il parametro body che contiene i valori da inviare al server
- recuperare la risposta con la proprietà responseText o responseXML

Il flusso di una richiesta asincrona con XMLHttpRequest è:

- creare un istanza dell'oggetto XMLHttpRequest
- assegnare una funzione di callback attraverso la proprietà onreadystatechange
- invocare il metodo open con i parametri metodo, url e il valore false per qualificare le richiesta come sincrona
- se la richiesta è di tipo POST utilizzare setRequestHeader per impostare l'header HTTP "Content-Type" sul valore "application/x-www-form-urlencoded"
- invocare il metodo send con il parametro body che contiene i valori da inviare al server

- la funzione di callback verrà chiamata ad ogni cambiamento di stato della richiesta, all'interno di questa funzione ci dovrà essere il controllo della proprietà readyState ed il recupero, una volta accertato che readyState sia 4 (completed), della risposta con la proprietà responseText o responseXML

Non solo XML

Una cosa importante da notare è che XMLHttpRequest può essere utilizzato per ottenere dal server non solo XML well-formed, ma qualunque tipo di file di testo : HTML, testo ASCII ecc…

I valori che può assumere readyState di XMLHttpRequest sono:

(0) UNINITIALIZED	L'oggetto è stato creato, ma non inizializzato(open non è stata invocata).
(1) LOADING	L'oggetto è stato creato, ma send non è stato invocato.
(2) LOADED	send è stato invocato, ma la risposta non è ancora disponibile.
(3) INTERACTIVE	Alcuni dati sono stati ricevuti. La proprietà responseText contiene parte dei dati.
(4) COMPLETED	Tutti i dati sono stati ricevuti e sono disponibili attraverso responseText o responseXML.

I codici di status HTTP e i relativi valori di statusText (almeno quelli più probabili) che possiamo ricevere dal server sono:

status	statusText
Success	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status

status	statusText
Redirection	
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
Errors	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
509	Bandwidth Limit Exceeded

Nota importante

Le versioni attuali di XMLHttpRequest non consentono di connettersi a domini differenti rispetto a quello dove si trova lo script che instanzia l'oggetto, per cui se lo script si trova, ad esempio, in http://www.mioserver.com/client.html potrà richiedere dati solo ad una pagina o procedura che risiede nello stesso dominio (come http://www.mioserver.com/server.aspx) e non in altri (come http://www.altroserver.com/server.aspx).

Si afferma che questa limitazione sia dovuta a motivi di sicurezza

(anche se non si è ben capito quali possano essere).
Se quella di connettersi a server esterni fosse proprio un'esigenza indispensabile (come il caso in cui sia necessario connettersi a un Web Service esterno) l'unica strada è quella di creare una pagina con un linguaggio lato server che faccia da "proxy": accetti cioè la richiesta dal client, la rigiri al server effettivo, recepisca la risposta e la reindirizzi nuovamente al client.
Un esempio di questa tecnica in PHP è illustrato su http://developer.yahoo.com/javascript/howto-proxy.html .

3.6 XMLHTTPREQUEST E JAVASCRIPT

Per fortuna con XMLHttpRequest non ci sono tutti i problemi che abbiamo visto con DOM e XPath tra le diverse versioni dei browser. Le uniche differenze sono tra Internet Explorer 5 e 6 e gli altri browser (fra cui Internet Explorer 7), i primi implementano XMLHttpRequest come oggetto ActiveX, mentre gli altri lo includono nel motore del browser.

Internet Explorer 5 e 6

L'istanza di un oggetto XMLHttpRequest si effettua con il seguente codice:

```
var req = new ActiveXObject ("Msxml2.XMLHTTP");
```

Internet Explorer 7, Firefox e altri

L'istanza è effettuata direttamente con l'operatore new

```
var req = new XMLHttpRequest()
```

Una volta instanziata una XMLHttpRequest il funzionamento è identico per tutti i browser.
Vediamo quindi come gestire una richiesta sincrona in POST:

```
function getXMLHttpRequest (){
        if (typeof(XMLHttpRequest) != 'undefined') {
                return new XMLHttpRequest();
        }
else {
                return new ActiveXObject("Msxml2.XMLHTTP");
}
}

function sendSync (){
        var req = getXMLHttpRequest();
        req.open("POST", "data.xml", false);
        req.setRequestHeader ("Content-Type", "application/x-www-form-
urlencoded");
        req.send(" ");
        return req.responseText;
}
```

e qui invece abbiamo una richiesta asincrona (la funzione getXMLHttpRequest sarà la stessa dell'esempio precedente):

```
function sendAsync () {
        var req = getXMLHttpRequest();
        var onreadystate = function (){
                if (req.readyState == 4) {
                        if (req.status == 200) {
                                var risposta = req.responseText;
                                //fare qualcosa con la risposta
                        }
                        else {
                                var error = req.statusText;
                                //si è verificato un errore
                        }
```

```
            }
            else {
                  //in attesa
            }
      }
      req.onreadystatechange = onreadystate; //assegnazione al callback
                                                                                interno
      req.open("POST", "data.xml", true);
      req.setRequestHeader ("Content-Type",
                                    "application/x-www-form-urlencoded");
      req.send(" ");
}
```

si noti come abbiamo mantenuta compatta la funzione inserendo il callback come funzione interna.

Invio di parametri al server

Negli esempi precedenti nel metodo send abbiamo indicato una stringa vuota, questo perché si trattava semplicemente di scaricare un documento statico (il file data.xml), tuttavia in applicazioni AJAX la fonte XML sarà probabilmente generata dinamicamente dal server e quindi avremo bisogno di passargli dei parametri (ad esempio per inviargli i dati di una form) questi parametri devono essere concatenati in una stringa simile a quella che viene usata per le URL e cioè:

```
<nome>=<valore>&<altro_nome>=<altro_valore>
```

In javascript potremmo concatenare e inviare i parametri con:

```
var params = "nome=Mario&cognome=Rossi&indirizzo=via verdi";
req.send(params);
```

Nel caso di variabili conviene però codificare i valori con encodeU-RIComponent per evitare caratteri riservati :

```
function sendData (nome,cognome,indirizzo) {
...
var params =  "nome=" + encodeURIComponent(nome) +
      "&cognome=" + encodeURIComponent(cognome) +
      "&indirizzo=" + encodeURIComponent(indirizzo);
req.send(params);
}
```

3.7 UNA LIBRERIA JAVASCRIPT PER DOM, XPATH, XMLHTTPREQUEST

A questo punto abbiamo tutti gli strumenti concettuali per gestire l'XML, però non possiamo certo lavorare con due approcci completamente diversi all'XML come quelli offerti da Internet Explorer e Firefox; a parte l'oggettiva difficoltà di destreggiarsi tra le varie sintassi, c'è anche da considerare che il nostro codice sarebbe disseminato di if ed estremamente prolisso.
Quello che ci serve, a questo punto, è una libreria javascript intermedia che si occupi di offrire modalità di accesso uniche a XPath, DOM e XMLHttpRequest.

3.8 COS'È UNA LIBRERIA JAVASCRIPT?

Per chi non lo sapesse, una libreria javascript non è altro che un (o più) file che contengono codice javascript con funzioni di supporto. Nella pagina Web gli script presenti possono utilizzare le funzioni di una libreria inserendo uno script di riferimento al file che la contiene con :

```
<script src="libreria.js" type="text/javascript"></script>
```

in tal modo tutte le funzioni contenute nella libreria vengono rese disponibili agli script della pagina.
Potete provare anche voi a fare la vostra libreria, scrivete del codice in un file che chiamerete ad esempio libreria.js (tipicamente il file di libreria ha estensione .js ma questa è solo una convenzione tacita, non una regola), ad esempio:

```
function metodoLibreria(messaggio){
      alert(messaggio)
}
```

Ricordate, nei file di libreria non devono essere inclusi i TAG <script>, solo il codice javascript.
Nella pagina HTML di destinazione sarà quindi possibile fare riferimento alla libreria:

```
<html>
<head>
      <script language="javascript" type="text/javascript"
                                        src="libreria.js"></script>
</head>
<body></body>
</html>
```

a questo punto qualsiasi script della pagina potrà utilizzare la funzione metodoLibreria :

```
<html>
<head>
      <script language="javascript" type="text/javascript"
                                        src="libreria.js"></script>
```

```
    <script language="javascript" type="text/javascript">
        metodoLibreria ("Ciao a tutti!");
    </script>
</head>
<body>
</body>
</html>
cioè sarebbe stato un po’ come scrivere:
<html>
<head>
    <script language="javascript" type="text/javascript">
    function metodoLibreria(messaggio){
        alert(messaggio)
    }
    </script>
    <script language="javascript" type="text/javascript">
        metodoLibreria ("Ciao a tutti!");
    </script>
</head>
<body>
</body>
</html>
```

per chi è pratico di programmazione web lato server è un po’ come quando si usa una direttiva include per far riferimento a codice che è comune a più pagine.

3.9 CERCASI LIBRERIA JAVASCRIPT PER XML…

In rete ci sono diverse librerie javascript che fanno da Wrapper nei confronti dei vari browser, cioè in pratica forniscono delle funzioni che internamente chiamano quelle implementate dal browser.

Per essere ancora più chiari: abbiamo visto che l'istanza di un documento DOM si crea con:

Internet Explorer

```
var objDOM = new ActiveXObject("Msxml2.DOMDocument");
```

Firefox

```
var objDOM = document.implementation.createDocument("", "", null)
```

Una libreria wrapper offrirà quindi un metodo unico per creare un'istanza DOM, tipo:

```
var objDOM = libreria.getDomDocument();
```

naturalmente la libreria svilupperà, al suo interno una funzione getDomDocument che potrebbe essere (semplificando molto) un po' come questa:

```
libreria.getDomDocument = function (){
    if(isIE) {
        return new ActiveXObject("Msxml2.DOMDocument");
    }
    else {
        return document.implementation.createDocument("", "", null);
    }
}
```

cioè in pratica effettua il riconoscimento del browser e restituisce l'oggetto creato con il metodo nativo.
Il vantaggio di questa tecnica è evidente: poter utilizzare metodi univoci senza preoccuparsi dell'implementazione del browser sotto-

stante.

Sarissa

Una delle più diffuse librerie wrapper è Sarissa (il nome deriverebbe da una lunga lancia usata dalle falangi Macedoni) che è scaricabile all'indirizzo https://sourceforge.net/projects/sarissa .
Questa libreria offre appunto una serie di funzioni per XML che si sostituiscono a quelle del browser. Vediamo quali.

Creazione di un'istanza DOM XML

È sufficiente richiamare il metodo:

```
var objDom = Sarissa.getDomDocument();
```

è possibile chiamare questa funzione anche con due parametri che rappresentano il default namespace e il nome dell'elemento radice:

```
var objDom = Sarissa.getDomDocument("http://xml.org/test","root");
creando così in memoria un documento XML tipo:
<root xmlns="http://xml.org/test"></root>
```

Caricare un documento DOM XML da una stringa

Si utilizzerà in questo caso l'oggetto DOMParser di Sarissa in questo modo:

```
var objDom = Sarissa.getDomDocument();
var xmlString = "<root>my xml!</root>";
objDom = (new DOMParser()).parseFromString(xmlString, "text/xml");
```

Creare un'istanza di XMLHttpRequest

L'istanza viene richiamata semplicemente attraverso:

```
var xmlhttp = new XMLHttpRequest();
```

Utilizzando poi l'oggetto come abbiamo visto in precedenza.

Caricare un documento DOM XML da un file

È possibile effettuare il caricamento sincrono o asincrono del file sebbene sia disponibile anche il metodo load per DOM XML che emula quello di Microsoft esso è deprecato dagli autori e potrebbe non essere presente nelle future versioni, essi consigliano di usare XMLHttpRequest.
In modo **sincrono** sarà quindi:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "data.xml", false);
xmlhttp.send('');
var objDom = xmlhttp.responseXML;
e in modo asincrono:
function loadDoc(){
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.open("GET", "data.xml", true);
    xmlhttp.onreadystatechange = myHandler;
    xmlhttp.send('');
    function myHandler(){
        if(xmlhttp.readyState != 4) return ;
        var objDom = xmlhttp.responseXML;
    }
}
```

Ottenere una stringa da un nodo XML

A volte può essere necessario, soprattutto in fase di debug, controllare i dati ricevuti e trasformare quindi il documento XML in una stringa, magari da visualizzare con un'alert.
Per svolgere questo compito Sarissa prevede :

```
var strXml = new XMLSerializer().serializeToString(objDom)
```

Effettuare una ricerca XPath

Per XPath Sarissa ricalca il modello usato da Microsoft, abilitando i metodi selectNodes e selectSingleNode anche per gli altri browser, per cui la ricerca si presenta semplice:

```
...
var objDom = xmlhttp.responseXML;
var list = objDom.selectNodes ("//persona");
s = "";
for(var i=0;i<list.length;i++){
        s += list[i].getAttribute("nome") + "\r\n";
}
alert(s);
```

jsXML

Sarissa è una bellissima libreria, con il vantaggio di essere anche molto diffusa, ci sono però alcuni svantaggi:

- Alcuni metodi (vedi il caricamento di file esterni) sono un po' prolissi e quindi richiederebbero ulteriori funzioni per semplificarne l'utilizzo.
- L'insieme del gruppo di librerie di Sarissa ha una dimensione di circa 60Kb, il che non è molto in senso assoluto, tuttavia può diventare un problema in applicazioni che già hanno altre librerie da includere; bisogna infatti ricordarsi che anche i file di libreria, come i CSS, le immagini ecc… contribuiscono a far lievitare il "peso" della pagina che il client deve scaricare portando facilmente una pagina a "pesare" 200/300 Kb, ricordatevi che non tutti hanno l'ADSL…

Queste considerazioni mi hanno portato a sviluppare una libreria personale per AJAX che ho chiamato jsXML, scaricabile dal sito: http://www.smelzo.it .

I principi su cui si basa questa libreria sono:

- Semplificare il più possibile le cose

- Ridurre le dimensioni (attualmente 11Kb)

Per arrivare alla semplificazione si è scelto la strada di emulare, anche per Firefox, il modello proposto da Microsoft, è vero che quest'ultimo non è il massimo di aderenza agli standard del W3C tuttavia lo ritengo estremamente più intuitivo e sintetico.
Sempre nell'ottica della semplificazione, si sono implementati dei metodi che consentono di fare delle operazioni con una sola riga di codice.
Alla riduzione (ad un sesto di Sarissa) si è arrivati invece con la scelta di limitarsi alla compatibilità con Internet Explorer e Firefox (molte funzioni lavorano tuttavia correttamente anche in Opera); questo può anche sembrare poco "politically correct" tuttavia c'è da considerare IE e Firefox insieme coprono ormai quasi il 98% del mercato e praticamente tutti i sistemi operativi e che le applicazioni AJAX vengono usate soprattutto in parti del sito "controllate" come forum, amministrazione ecc… dove cioè si ha un certo "potere" di indirizzamento verso l'uso di un certo browser.
Vediamo quindi come compiere le più comuni operazioni con XML con jsXML.

Creazione di un'istanza DOM XML

Una nuova istanza si ottiene semplicemente con :

```
var objDom= jsXML.newDOMDocument();
```

Caricare un documento DOM XML da una stringa

Anche qui si semplificano le cose riunendo creazione di un'istanza e caricamento da stringa in un'unica funzione:

```
var objDom= jsXML.parseXMLDocument(strXml);
```

dove il parametro sarà ovviamente la stringa XML su cui costruire il nuovo documento

Gestione XMLHttpRequest

Essendo un'operazione frequentissima in AJAX le operazioni XMLHttpRequest sono completamente gestite da funzioni di utilità.
Per una richiesta **sincrona** :

```
var strXML = jsXML.sendSync ("data.xml","param1=X&param2=Y");
```

in questo, come in altri casi la richiesta dà una risposta in forma di stringa e non di XML per permettere di gestire anche risorse non XML.
Il primo parametro è l'URL della risorsa e il secondo sono i valori da inviare eventualmente al server , di default la richiesta è effettuata con POST ma si può utilizzare anche un altro metodo specificandolo come terzo parametro:

```
var strXML = jsXML.sendSync
                                    ("data.xml","param1=X&param2=Y","GET");
```

Nel caso che invece di una risposta come stringa si desideri ottenere direttamente una risposta come oggetto DOM XML è sufficiente usare jsXML.direct:

```
var objDOM = jsXML.direct ("data.xml","param1=X&param2=Y");
```

Per la richiesta **asincrona** il metodo base è:

```
jsXML.sendAsync ("POST", "data.xml","param1=X&param2=Y",
successHandler,errorHandler,waitHandler);
//callback Success
function successHandler(strXML) {
 // fai qualcosa con strXML
}
//callback Error
```

```
function errorHandler (statusText) {
 // fai qualcosa con statusText
}
//callback waitHandler
function errorHandler (readyState) {
 // fai qualcosa con readyState
}
```

come avrete intuito, il primo parametro è il metodo utilizzato, il secondo e il terzo sono URL e valori da inviare e i restanti tre sono riferimenti a funzioni di callback : la prima viene invocata in caso di successo e riceve come parametro la risposta come stringa XML, la seconda viene invocata in caso di errore e riceve come parametro il messaggio di errore http, l'ultima viene invocata mentre si attende la risposta e può essere utile per mostrare un messaggio di attesa.
Dei tre callback è obbligatorio solo il primo.
Per semplificare ulteriormente si hanno anche altre due funzioni satellite che indirizzano direttamente verso la chiamata GET o POST eliminando il primo parametro:

```
jsXML.sendAsyncGet ("data.xml","param1=X&param2=Y",
successHandler,errorHandler,waitHandler);
```

e

```
jsXML.sendAsyncPost ("data.xml","param1=X&param2=Y",
successHandler,errorHandler,waitHandler);
```

Caricare un documento DOM XML da un file

Si riproduce completamente il metodo load di Microsoft per cui è sufficiente:

```
var objDom= jsXML.newDOMDocument();
objDom.load ("data.xml");
```

Addirittura c'è una funzione (openDOMDocument) che fa tutto in un colpo solo:

```
var objDOM = jsXML.openDOMDocument("data.xml");
```

Ottenere una stringa da un nodo XML

Anche qui si è seguita la strada di Microsoft per cui per ogni nodo documento o elemento è sufficiente richiamare la proprietà xml:

```
var strXML = objDom.xml; //per l'intero documento
var strXML = objDom.documentElement.xml; //per un altro nodo
```

Effettuare una ricerca XPath

Come in Sarissa si sono collegate le query XPath ai metodi selectNodes e selectSingleNode e quindi:

```
var objDom= jsXML.newDOMDocument();
objDom.load ("data.xml");
var list = objDom.selectNodes ("//persona");
s = "";
for(var i=0;i<list.length;i++){
	s += list[i].getAttribute("nome") + "\r\n";
}
alert(s);
```

google-ajaxslt

Un'altra libreria per AJAX è google-ajaxslt (http://code.google.com/p/ajaxslt), la citiamo più per l'approccio particolare che adotta che per la reale praticabilità.
Il progetto è un'implementazione di XPath e XSLT fatta completamente in javascript! Cioè un'impresa a dir poco "titanica".
La scelta dei progettisti di questa libreria è quella di non appoggiarsi per nulla alle implementazioni native di DOM XML, XPath e XSLT

ma di riscriverli da zero (e per giunta in javascript!) eliminando così alla radice il problema delle differenze tra i vari browser.
Agli scopi pratici la libreria non è ancora ad uno stadio "maturo", un po' perché mancano una serie di funzioni indispensabili, un po' perché più orientata a XSLT che in AJAX può sì avere un ruolo, ma non proprio di primo piano.
L'idea infatti sarebbe di trasformare l'input XML ricevuto dal server, attraverso XSLT, in codice HTML, questo approccio può andare bene per del contenuto statico, ma se nella fase intermedia tra acquisizione e creazione del contenuto si dovessero impostare elementi HTML con gestori di eventi questo approccio non andrebbe più bene.
Comunque il progetto è veramente notevole per lo stile e l'abilità di scrittura, una vera miniera di spunti; i programmatori un po' più esperti di javascript farebbero bene a dare almeno un'occhiata ai sorgenti.

Microsoft AJAX Library

Tutt'altra strada è quella intrapresa dalla Microsoft AJAX Library, la libreria javascript client-side che è parte del più ampio progetto ASP.NET AJAX (http://ajax.asp.net). Sebbene sia concepita per essere la parte client di un sistema che prevede l'integrazione di alcuni nuovi Web Controls per ASP.NET (ne parliamo più avanti nel capitolo "Finalmente AJAX"), la libreria può essere utilizzata anche autonomamente da ASP.NET cioè anche con altri linguaggi.
La sua filosofia è un po' quella di Sarissa, ovvero creare un wrapper delle funzioni native per XML dei vari browser in circolazione, tuttavia per farlo mette in piedi una gigantesca API dove c'è addirittura un tentativo di tipizzazione di un linguaggio di scripting, per natura non tipizzato, come javascript.
Il risultato è indubbiamente molto Object Oriented, tuttavia se lo scopo era quello di semplificare le cose sembra proprio che siamo lontani dall'obbiettivo.

Il lato positivo di Microsoft AJAX Library è che funziona anche con Opera e Safari, ma allora – secondo me – conviene usare Sarissa che ha un'API decisamente più intuitiva.

3.10 METTIAMO TUTTO INSIEME

Con quello che abbiamo appreso su XML e la sua gestione con javascript torniamo adesso all'esercizio che abbiamo visto nel primo capitolo affrontando lo stesso problema con la differenza che questa volta dovremo riuscire a caricare i dati dal file XML esterno.

Problema

Creare una pagina HTML con un BUTTON e un DIV vuoto.
Sull'evento click associato al BUTTON si deve compiere questa azione:

- Se il DIV è vuoto creare una TABLE, leggendo i valori dal file esterno data.xml presente nella stessa directory, e inserirla sotto di esso
- Se il DIV è nascosto visualizzarlo, viceversa se è visibile nasconderlo

Soluzione

Passo 1 – creare la pagina HTML

In un editor di testo creiamo la pagina HTML che abbiamo utilizzato in precedenza, ma con l'aggiunta di un riferimento ad una libreria di funzioni per XML (in questo caso jsXML) :

```
<html>
  <head>
      <title>Esempio</title>
     <script language="javascript" type="text/javascript"
                                                src="jsXML.js"></script>
     <script language="javascript" type="text/javascript">
           //codice javascript
```

```
    </script>
  </head>
  <body>
    <button id="btn1" onclick="cliccami()">Clicca qui per vedere la
    tabella!</button>
    <div id="container" style="display:none"></div>
  </body>
</html>
```

Passo 2 – codice DOM per la creazione della tabella

Questa è la funzione che subisce maggiori cambiamenti perché questa volta la sorgente dei dati è un file XML che dovrà essere caricato e letto con XPath:

```
function createTable(){
      //definizione dati tabella
      //prima riga
      var headers = ["nome","cognome","indirizzo"];
      var objDOM = jsXML.openDOMDocument("data.xml");
      var nodes = objDOM.selectNodes("//persona");
      var oTable = document.createElement("TABLE");
      //proprietà tabella
      oTable.width = "400";
      oTable.cellSpacing = "0";
      oTable.cellPadding = "2";
      //prima riga
      var oTr = oTable.insertRow(-1);
      for(var i=0;i<headers.length;i++){
            var oTd = oTr.insertCell(-1);
            oTd.align="center";
            oTd.background = "#EEEEEE";
            oTd.style.font = "bold 12px arial";
            oTd.style.borderBottom = "1px solid #666666";
```

```
            oTd.innerHTML= headers[i];
      }
      //righe dati
      for(var i=0;i<nodes.length;i++){
      var node = nodes[i];
      var oTr = oTable.insertRow(-1);
            for(var n=0;n<headers.length;n++){
                  var oTd = oTr.insertCell(-1);
                  oTd.style.font = "12px verdana";
                  oTd.style.borderBottom = "1px solid #999999";
                  oTd.innerHTML= node.getAttribute(headers[n]) ;
            }
      }
      currentTable = oTable;
      var container = document.getElementById("container");
      container.appendChild(currentTable);
}
```

in pratica :

- apriamo il documento XML
- estraiamo con XPath i nodi <persona>
- nel ciclo di costruzione della tabella leggiamo i vari attributi con getAttribute

le altre parti del programma sono uguali a quanto visto nel primo capitolo e anche il risultato è il medesimo.

LATO SERVER

Dopo aver trattato il più diffusamente possibile le problematiche client-side legate a javascript, HTML DOM e XML, dobbiamo adesso prendere in esame l'altro importante attore di AJAX: il lato server.
Infatti, pur ridotto nell'importanza rispetto alle Web Applications tradizionali, il server continua ad avere una funzione basilare : quella di sorgente di dati.

In AJAX, come abbiamo detto, abbiamo un colloquio ben definito tra gli script che ci sono nella pagina, un ciclo molto simile per certi aspetti a quello delle applicazioni client/server tradizionali:

1. client‡server – richiesta dati
2. server‡client – invio dati
3. client – elaborazione dati e cambiamenti all'interfaccia utente

Da quanto abbiamo visto finora con XMLHttpRequest dovremo essere in grado di inviare una richiesta al server e quindi assolvere al punto 1. Ma siamo anche in grado di manipolare adeguatamente con javascript sia un documento XML (con il DOM e XPath) che l'output HTML (con il DOM HTML e i CSS) e quindi siamo pronti anche per il punto 3.
Quello che ci manca è quindi soltanto il punto 2 : costruire una risposta dal lato server (che, ovviamente, dovrà essere un flusso XML).
Il 99% delle operazioni che restano da fare al server riguardano i dati (che siano database, file XML o file di testo non ha importanza) e sono, tipicamente:

- selezione
- inserimento
- modifica
- cancellazione

Ogni sviluppatore avrà naturalmente il suo linguaggio preferito per compiere queste operazioni lato server, noi partiremo comunque da un problema specifico da risolvere per vedere come implementare soluzioni in diversi linguaggi, questo naturalmente è solo un esempio, in applicazioni

reali le cose potrebbero essere più complesse, tuttavia negli esempi che faremo cercheremo di toccare le problematiche principali.

4.1 IL PROBLEMA

Abbiamo un database, il buon vecchio Northwind il database di esempio della Microsoft, che si trova in tutte le edizioni di Access e anche in qualche distribuzione di SQL Server; per collegarsi a Northwind utilizzeremo ODBC (in modo da mantenersi il più neutrali possibile rispetto al tipo di database di origine), vogliamo selezionare i dati dalla tabella Customers (clienti).
Questa selezione sarà influenzata, nella quantità di dati restituiti, dall'input dell'utente che potrà:

- impostare una stringa di ricerca
- impostare il campo dove effettuare la ricerca

Si tratta quindi di predisporre una pagina server-side che si connetta al Database ODBC, trasmetta la query costruita in base alle impostazioni dell'utente, recuperi i dati e li trasmetta come response XML.
La connessione ODBC avrà come nome NWIND.

4.2 FACCIAMOLO IN PHP

Creiamo sul server un file che chiameremo server.php, il file sarà solo codice (non conterrà quindi HTML) e cominciamo con la connessione.

La connessione

Il meccanismo di connessione sfrutta le funzioni ODBC integrate nel linguaggio e segue più o meno questo schema:

```
//connessione al database
$connectionstring = odbc_connect("NWIND", "xx", "xx");

//SQL query
```

```
$Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
                                                              Customers ";

//recupero dati
$queryexe = odbc_do($connectionstring, $Query);

//lettura dati
while(odbc_fetch_row($queryexe)) {
   //lettura campi
   $cname = odbc_result($queryexe, "CompanyName");
...
      }

//disconnessione dal database
odbc_close($connectionstring);
```

quindi :

- connessione
- impostazione query
- esecuzione query
- lettura di righe e campi
- disconnessione

Fin qui tutto bene, ma bisogna ricordarsi che l'output dovrà essere in XML e quindi dovremo impostare l'Header di risposta con il Content-Type appropriato ovvero :

```
header('Content-type: text/xml');
```

Costruire la query

C'è poi da gestire la query impostando la selezione in base dell'input dell'utente.

Partiamo proprio dall'input dell'utente, per prima cosa dovremo intercettare le variabili Request che potranno essere :

- search – la stringa da cercare
- field – il campo in cui effettuare la ricerca

Prevederemo anche il caso in cui l'utente non imposti il campo valorizzando la variabile con il nome del primo campo della query.
Il tutto si traduce nel codice:

```
$search = $_REQUEST["search"];
$field = $_REQUEST["field"];
if($field=="") $field = "CompanyName";
```

Occorre poi utilizzare questi parametri per costruire la query:

```
//SQL query
$Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
                                                                  Customers ";
if ($search!='') {
      $search = preg_replace("/'+/","''",$search);
$Query .= " WHERE $field LIKE '$search%'";
      }
```

cioè : se è presente l'impostazione della variabile $search ripuliamo la stringa dal carattere "'" che darebbe luogo a errori SQL sostituendolo con i doppi apici; costruiamo poi la clausola WHERE accodandola alla stringa $Query.

Costruire la risposta XML

L'XML sarà costruito all'interno del ciclo di lettura dei dati come stringa e restituito con echo:

```
$s = "<root>";
//lettura dati
while(odbc_fetch_row($queryexe)) {
      $s .= "<persona";
```

```
$cname = enc(odbc_result($queryexe, "CompanyName"));
        $pname = enc(odbc_result($queryexe, "ContactName"));
        $address = enc(odbc_result($queryexe, "Address") . " " .
        odbc_result($queryexe, "City") . " " . odbc_result($queryexe, "Country")
                                                                        );
        $s .= " company=\"$cname\"";
        $s .= " contact=\"$pname\"";
        $s .= " address=\"$address\"/>";
        }
$s .= "</root>";
//disconnessione dal database
odbc_close($connectionstring);
echo($s);
```

avrete senz'altro notato che il risultato della lettura del campo, effettuato con la funzione odbc_result, è stato passato come parametro alla funzione enc?
Abbiamo fatto in questo modo perché nella stringa proveniente dal database ci potrebbero essere dei caratteri speciali che XML potrebbe segnalare come errori.
La funzione enc si occupa appunto di codificare i caratteri della stringa in formato comprensibile per XML:

```
function enc($s){
        return xmlentities($s);
}
```

La funzione di appoggia a sua volta alla funzione xmlentities (presa pari pari da internet su http://theserverpages.com/php/manual/en/function.htmlentities.php) che è espressa così:

```
function xmlentities($string, $quote_style=ENT_QUOTES){
   static $trans;
```

```
  if (!isset($trans)) {
      $trans = get_html_translation_table(HTML_ENTITIES, $quote_style);
      foreach ($trans as $key => $value)
          $trans[$key] = '&#'.ord($key).';';
      $trans[chr(38)] = '&';
  }
      return  preg_replace("/&(?![A-Za-z]{0,4}\w{2,3};|#[0-9]{2,3};)/","&#38;" ,
strtr($string, $trans));
}
```

Il risultato

Il risultato finale del nostro lavoro sarà quindi:

```
<?php

header('Content-type: text/xml');

$search = $_REQUEST["search"];
$field = $_REQUEST["field"];
if($field=="") $field = "CompanyName";
//connessione al database
$connectionstring = odbc_connect("NWIND", "fs", "fs");

//SQL query
$Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
                                                            Customers ";
if ($search!='') {
      $search = preg_replace("/'+/","''",$search);
$Query .= " WHERE $field LIKE '$search%'";
      }
//recupero dati
$queryexe = odbc_do($connectionstring, $Query);
$s = "<root>";
```

```
//lettura dati
while(odbc_fetch_row($queryexe)) {
      $s .= "<persona";
      $cname = enc(odbc_result($queryexe, "CompanyName"));
   $pname = enc(odbc_result($queryexe, "ContactName"));
   $address = enc(odbc_result($queryexe, "Address") . " " .
   odbc_result($queryexe, "City") . " " . odbc_result($queryexe, "Country") );
   $s .= " company=\"$cname\"";
   $s .= " contact=\"$pname\"";
   $s .= " address=\"$address\"/>";
      }
$s .= "</root>";
//disconnessione dal database
odbc_close($connectionstring);
echo($s);

/*funzioni di encode XML*/
function xmlentities($string, $quote_style=ENT_QUOTES)
{
  static $trans;
  if (!isset($trans)) {
      $trans = get_html_translation_table(HTML_ENTITIES, $quote_style);
      foreach ($trans as $key => $value)
          $trans[$key] = '&#'.ord($key).';';
      $trans[chr(38)] = '&';
  }
  return preg_replace("/&(?![A-Za-z]{0,4}\w{2,3};|#[0-9]{2,3};)/","&#38;" ,
                                                        strtr($string, $trans));
}
function enc($s){
      return xmlentities($s);
}
?>
```

Puntiamo quindi il browser sulla pagina in questione e otterremo l'elenco completo della query sulla tabella in formato XML (vedi figura 3).

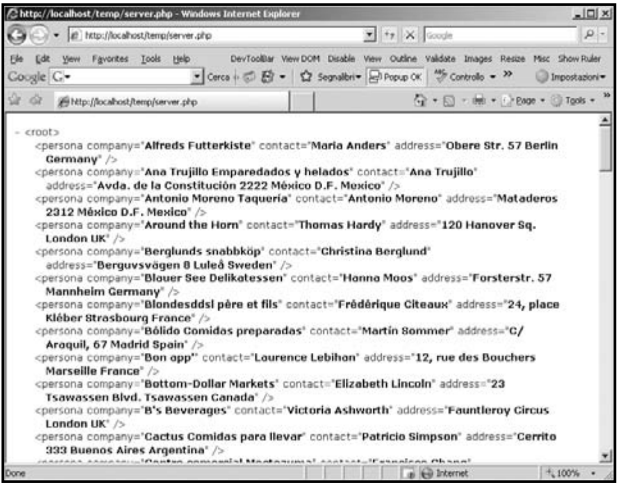


Figura 3: La query in XML da php

A questo punto potremo sbizzarrirci con le prove di selezione impostando nella URL i vari parametri di selezione; ad esempio scrivendo la URL come

```
http://mioserver/server.php?search=b&field=contactName
```



Figura 4: la query in XML da php con selezione

Avremo sul browser il risultato di cui in figura 4 (ovviamente la parte di URL http://mioserver andrà sostituita con l'indirizzo del server in cui avrete pubblicato la pagina)

4.3 FACCIAMOLO IN CLASSIC ASP

Vediamo adesso come risolvere il problema in classic ASP, a qualcuno sembrerà di fare "archeologia" visto che siamo quasi arrivati alla versione 3 del .NET Framework ma il vecchio ASP è ancora molto usato accingiamoci quindi a scrivere in un file che chiameremo server.asp.
La dinamica della connessione in ASP è la seguente:

```
'stringa di connessione
connString ="DSN=NWIND;Uid=fs;Pwd=fs;"
Dim conn , rs
'istanza di connessione
set conn=CreateObject("ADODB.CONNECTION")
conn.ConnectionString=connString
'istanza di recordset
set rs = CreateObject("ADODB.RECORDSET")
'Query
Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
Customers "
'apertura di connessione e recordset
conn.Open
rs.Open Query,conn
do until rs.EOF
'ciclo di lettura delle righe
rs.MoveNext
loop
'chiusura e rilascio oggetti
rs.Close
conn.Close
```

```
set rs = nothing
set conn = nothing
```

Anche in ASP dovremo impostare poi il ContentType della risposta su XML

```
Response.ContentType= "text/xml"
```

Costruire la query

Valutiamo l'input dell'utente contenuto nelle variabili search e field e costruiamo la query di selezione :

```
Dim search,field
search = Request("search")
field  = Request("field")
if field="" then field = "CompanyName"
...
Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
                                                                  Customers "
if search<>"" then
      search = replace(search,"'","''")
      Query = Query & " WHERE " & field & " LIKE '" & search & "%'"
end if
```

La logica non cambia rispetto all'esempio in PHP.

Costruire l'XML

La parte più difficile è la costruzione dell'XML, se avessimo utilizzato un documento DOM XML le cose sarebbero state semplici : istanza dell'oggetto, costruzione dei nodi e impostazione degli attributi. Il problema è che l'implementazione del DOM utilizzabile da ASP non effettua la conversione corretta delle stringhe di caratteri non ASCII, siamo quindi costretti a usare SAX per produrre XML, questo comunque è anche un vantaggio perché SAX è molto più veloce nella risposta.

Quindi vediamo come è gestito l'output XML con SAX:

```
Set xmlWriter = Server.CreateObject("MSXML2.MXXMLWriter.4.0")
xmlWriter.encoding="UTF-8"
'invia output del writer al response stream di ASP
xmlWriter.output = Response
'Genera eventi SAX
xmlWriter.startDocument
xmlWriter.startElement "", "", "root", nothing
Set objAttributes = Server.CreateObject("Msxml2.SAXAttributes.4.0")
do until rs.EOF
    cname = rs("CompanyName")
    pname = rs("ContactName")
    address = rs("Address") & " " & _
              rs("City") & " " & _
              rs("Country")
    objAttributes.addAttribute "","","company","CDATA",CStr(cname)
    objAttributes.addAttribute "","","contact","CDATA",CStr(pname)
    objAttributes.addAttribute "","","address","CDATA",CStr(address)
    xmlWriter.startElement "", "", "persona", objAttributes
    xmlWriter.endElement "", "", "persona"
    objAttributes.clear
rs.MoveNext
loop
xmlWriter.endElement "", "", "root"
xmlWriter.endDocument
```

come possiamo notare SAX scrive direttamente sullo stream Response di ASP

Il risultato

Il risultato finale del nostro lavoro in classic ASP sarà quindi:

```
<%
Response.ContentType= "text/xml"
Dim search,field
Dim connString,Query
search = Request("search")
field  = Request("field")
if field="" then field = "CompanyName"
connString ="DSN=NWIND;Uid=fs;Pwd=fs;"
Dim conn , rs
set conn=CreateObject("ADODB.CONNECTION")
conn.ConnectionString=connString
set rs = CreateObject("ADODB.RECORDSET")

Query = "SELECT CompanyName, ContactName, Address, City, Country FROM
Customers "
if search<>"" then
	search = replace(search,"'","''")
	Query = Query & " WHERE " & field & " LIKE '" & search & "%'"
end if

conn.Open
rs.Open Query,conn
'Istanza di xmlWriter
Set xmlWriter = Server.CreateObject("MSXML2.MXXMLWriter.4.0")
xmlWriter.omitXMLDeclaration = True
xmlWriter.indent = false
xmlWriter.encoding="UTF-8"
'invia output del writer al response stream di ASP
xmlWriter.output = Response
'Genera eventi SAX
xmlWriter.startDocument
xmlWriter.startElement "", "", "root", nothing
Set objAttributes = Server.CreateObject("Msxml2.SAXAttributes.4.0")
```

```
do until rs.EOF
      cname = rs("CompanyName")
      pname = rs("ContactName")
      address = rs("Address") & " " & _
                    rs("City") & " " & _
                    rs("Country")
      objAttributes.addAttribute "","","company","CDATA",CStr(cname)
      objAttributes.addAttribute "","","contact","CDATA",CStr(pname)
      objAttributes.addAttribute "","","address","CDATA",CStr(address)
      xmlWriter.startElement "", "", "persona", objAttributes
      xmlWriter.endElement "", "", "persona"
      objAttributes.clear
rs.MoveNext
loop
xmlWriter.endElement "", "", "root"
xmlWriter.endDocument
rs.Close
conn.Close
set rs = nothing
set conn = nothing
set objAttributes = nothing
set xmlWriter= nothing
%>
```

Puntiamo il nostro browser sulla pagina server.asp del sito di test e otterremo come risultato lo stream XML della selezione effettuata (in maniera identica a quanto avveniva con PHP).
Anche qui naturalmente potremmo provare varie selezioni impostando i parametri della queryString della URL come ad esempio:

```
http://mioserver/server.asp?search=b&field=contactName
```

sempre naturalmente sostituendo la parte di URL http://mioserver con

l'indirizzo del server in cui avrete pubblicato la pagina.

4.4 FACCIAMOLO IN ASP.NET CON C#

Affrontiamo adesso lo stesso problema nell'ottica di ASP.NET con linguaggio C# , per la nostra pagina di risposta scegliamo il modello di scrittura di pagine ASP.NET inline, contrapposto al modello codebehind che prevede il codice in un file separato (qui sarebbe inutile visto che la pagina contiene solo codice).
Per prima cosa andiamo ad importare, nella nuova pagina server.aspx che scriveremo, gli spazi dei nomi che contengono le classi che ci serviranno per lavorare:

```
<%@ Page Language="C#"  %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Web" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.Odbc" %>
```

Scriveremo poi tutto il nostro codice all'interno del TAG <script>:

```
<script runat="server">
...
</script>
```

In C# possiamo modulare meglio il flusso del programma per cui impostiamo delle proprietà (Search e Field) che raccolgono i criteri di ricerca impostati dall'utente.

```
public string Search
  {
     get
```

```
    {
        string s = Request.Params.Get("search");
        if (!String.IsNullOrEmpty(s)) s = s.Replace("'", "''");
        return s;
    }
}
public string Field
{
    get
    {
        string f = Request.Params.Get("field");
        if (String.IsNullOrEmpty(f)) f = "CompanyName";
        return f;
    }
}
```

com'è possibile notare la gestione dei valori nulli e dei caratteri non validi avviene direttamente all'interno delle proprietà

Impostiamo quindi la funzione che compone la query al database:

```
private string GetQuery()
{
    string query = "SELECT CompanyName, ContactName, Address, City,
                                        Country FROM Customers ";
    if (Search != null)
    {
        query += String.Format(" WHERE {0} LIKE '{1}%'", Field, Search);
    }
    return query;
}
```

e, di seguito, quella che crea la connessione:

```
private OdbcConnection GetConn()
{
    string connStr = "DSN=NWIND;Uid=fs;Pwd=fs;";
    return new OdbcConnection(connStr);
}
```

Creazione dell'XML

Anche qui, come abbiamo fatto in classic ASP, utilizziamo un XmlWriter per scrivere i dati che provengono dal database, soltanto che in .NET l'oggetto è molto più semplice:

```
    XmlTextWriter writer = new XmlTextWriter(this.Response.Output);
    writer.WriteStartDocument();
    writer.WriteStartElement("root");
    while (reader.Read())
    {
        string cname = reader.GetString(0);
        string pname = reader.GetString(1);
        string address = reader.GetString(2) + " " + reader.GetString(3) + rea
                                                              der.GetString(4);
        writer.WriteStartElement("persona");
        writer.WriteAttributeString("company", cname);
        writer.WriteAttributeString("contact", pname);
        writer.WriteAttributeString("address", address);
        writer.WriteEndElement();
    }
    reader.Close();
    writer.WriteEndElement();
    writer.WriteEndDocument();
    writer.Close();
```

da notare che abbiamo puntato l'output dell'XmlWriter direttamente sul flusso della Response della pagina.

Il recupero dei dati, come in parte abbiamo visto, avviene per mezzo di un DataReader creato da un Command che esegue la query:

```
    OdbcConnection cnn = this.GetConn();
    OdbcCommand cmd = cnn.CreateCommand();
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = this.GetQuery();
    cnn.Open();
    OdbcDataReader reader =
               cmd.ExecuteReader(CommandBehavior.CloseConnection);
```

Il risultato

Tutto il codice della nostra pagina ASP.NET in C# sarà alla fine:

```
<%@ Page Language="C#"  %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Web" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="server">
  public string Search
  {
    get
    {
      string s = Request.Params.Get("search");
      if (!String.IsNullOrEmpty(s)) s = s.Replace("'", "''");
      return s;
    }
  }
  public string Field
  {
    get
```

```
        {
            string f = Request.Params.Get("field");
            if (String.IsNullOrEmpty(f)) f = "CompanyName";
            return f;
        }
    }

    private string GetQuery()
    {
        string query = "SELECT CompanyName, ContactName, Address, City,
                                                    Country FROM Customers ";
        if (Search != null)
        {
            query += String.Format(" WHERE {0} LIKE '{1}%'", Field, Search);
        }
        return query;
    }

    private OdbcConnection GetConn()
    {
        string connStr = "DSN=NWIND;Uid=fs;Pwd=fs;";
        return new OdbcConnection(connStr);
    }
    protected void Page_Load(object sender, EventArgs e) {
        this.Response.ContentType = "text/xml";
        OdbcConnection cnn = this.GetConn();
        OdbcCommand cmd = cnn.CreateCommand();
        cmd.CommandType = CommandType.Text;
        cmd.CommandText = this.GetQuery();
        cnn.Open();
        OdbcDataReader reader =
                          cmd.ExecuteReader(CommandBehavior.CloseConnection);
        XmlTextWriter writer = new XmlTextWriter(this.Response.Output);
```

```
        writer.WriteStartDocument();
        writer.WriteStartElement("root");
        while (reader.Read())
        {
            string cname = reader.GetString(0);
            string pname = reader.GetString(1);
            string address = reader.GetString(2) + " " + reader.GetString(3) +
                                                                    reader.GetString(4);
            writer.WriteStartElement("persona");
            writer.WriteAttributeString("company", cname);
            writer.WriteAttributeString("contact", pname);
            writer.WriteAttributeString("address", address);
            writer.WriteEndElement();
        }
        reader.Close();
        writer.WriteEndElement();
        writer.WriteEndDocument();
        writer.Close();
    }
</script>
```

Puntiamo il nostro browser sulla pagina server.aspx del sito di test e otterremo come risultato lo stesso XML degli esempi precedenti.
Anche qui naturalmente potremmo provare varie selezioni impostando i parametri della queryString della URL come ad esempio:

```
http://mioserver/server.aspx?search=a&field=Address
```

sempre naturalmente sostituendo la parte di URL http://mioserver con l’indirizzo del server in cui avrete pubblicato la pagina.

E se preferiamo VB …

Naturalmente l’esempio che abbiamo visto in C# può essere anche scrit-

to in VB.NET come:

```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Web" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.Odbc" %>
<%@ Page Language="VB" %>

<script runat="server">
    Public ReadOnly Property Search() As String
        Get
            Dim s As String = Request.Params.Get("search")
            If Not String.IsNullOrEmpty(s) Then s = s.Replace("'", "''")
            Return s
        End Get
    End Property
    Public ReadOnly Property Field() As String
        Get
            Dim s As String = Request.Params.Get("field")
            If Not String.IsNullOrEmpty(s) Then s = "CompanyName"
            Return s
        End Get
    End Property

    Private Function GetQuery() As String
        Dim query As String = "SELECT CompanyName, ContactName, Address,
                                                City, Country FROM Customers "
        If Not String.IsNullOrEmpty(Search) Then
            query &= String.Format(" WHERE {0} LIKE '{1}%'", Field, Search)
        End If
        Return query
    End Function
```

```
Private Function GetConn() As OdbcConnection
    Dim connStr As String = "DSN=NWIND;Uid=fs;Pwd=fs;"
    Return New OdbcConnection(connStr)
End Function

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
                                                  System.EventArgs)
    Me.Response.ContentType = "text/xml"
    Dim cnn As OdbcConnection = Me.GetConn()
    Dim cmd As OdbcCommand = cnn.CreateCommand()
    cmd.CommandType = CommandType.Text
    cmd.CommandText = Me.GetQuery()
    cnn.Open()
    Dim reader As OdbcDataReader =
                    cmd.ExecuteReader(CommandBehavior.CloseConnection)
    Dim writer As New XmlTextWriter(Me.Response.Output)
    writer.WriteStartDocument()
    writer.WriteStartElement("root")
    While reader.Read
        Dim cname As String = reader.GetString(0)
        Dim pname As String = reader.GetString(1)
      Dim address As String = reader.GetString(2) & " " & reader.GetString(3)
                                                    & reader.GetString(4)
        writer.WriteStartElement("persona")
        writer.WriteAttributeString("company", cname)
        writer.WriteAttributeString("contact", pname)
        writer.WriteAttributeString("address", address)
        writer.WriteEndElement()
    End While
    reader.Close()
    writer.WriteEndElement()
    writer.WriteEndDocument()
```

```
      writer.Close()
   End Sub
</script>
```

4.5 FACCIAMOLO CON UN WEB SERVICE .NET CON VB

Per completare la carrellata dei nostri esempi lato server terminiamo con la risoluzione del problema proposto all'inizio del capitolo attraverso un Web Service scritto con VB.NET.
Con .NET l'implementazione di un Web Service è veramente semplice, è sufficiente scrivere in un file con estensione .asmx una classe del tipo:

```
<%@ WebService Language="VB" Class="server" %>

<WebService()> _
Public Class server
   Inherits System.Web.Services.WebService
   ' Inserire codice qui …
End Class
```

Prima di iniziare dovremmo però definire un file web.config per l'applicazione Web. Il file web.config è un semplice file XML posto nella directory root del sito Web e contenente le impostazioni di configurazione.
Per i nostri scopi basta impostare il web.config in questo modo :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.web>
     <webServices>
       <conformanceWarnings>
         <clear/>
```

```
      </conformanceWarnings>
      <protocols>
        <add name="HttpPost"/>
        <add name="HttpGet"/>
      </protocols>
    </webServices>
  </system.web>
</configuration>
```

Quello che abbiamo visto serve per :

- Eliminare gli avvisi di conformità rispetto ad alcuni standard.
  Infatti poiché utilizziamo il flusso XML prodotto dal Web Service con AJAX, e quindi senza una deserializzazione, sarebbe superfluo gestire i Namespaces come invece richiederebbero gli standard di conformità.
- Abilitare l'utilizzo del Web Service da POST e/o da GET oltre che da SOAP AJAX attraverso il parsing XML potrebbe anche gestire SOAP, ma questo produrrebbe soltanto output più grandi, senza alcun beneficio pratico; quindi useremo POST, GET lo consentiamo per effettuare dei test, ma in produzione è bene toglierlo

L'output XML

Con un Web Service l'output è costituito sempre da XML e quindi non c'è alcuna necessità di controllare direttamente il flusso. Per trasformare l'oggetto di risposta in XML il Web Service utilizza la Serializzazione XML, quindi, per modellare la risposta secondo le nostre esigenze dovremo creare un oggetto con gli adeguati attributi di controllo della serializzazione.

Noi abbiamo quindi modellato il nostro oggetto in una classe:

```
  <XmlRoot("root", Namespace:="")> _
  Public Class Indirizzi
      Public Sub Add(ByVal company As String, ByVal contact As String, ByVal
                                                          address As String)
```

```
            Dim p As New Persona
            p.company = company
            p.contact = contact
            p.address = address
            Persone.Add(p)
        End Sub
        Private _Persone As List(Of Persona)
        <XmlElement("persona")> _
        Public Property Persone() As List(Of Persona)
            Get
                If _Persone Is Nothing Then
                    _Persone = New List(Of Persona)
                End If
                Return _Persone
            End Get
            Set(ByVal Value As List(Of Persona))
                _Persone = Value
            End Set
        End Property

        Public Class Persona
            <XmlAttribute()> Public company As String
            <XmlAttribute()> Public contact As String
            <XmlAttribute()> Public address As String
        End Class
    End Class
```

gli attributi <XmlRoot>, <XmlElement>, <XmlAttribute> permettono appunto di definire la formattazione in fase di serializzazione della classe.
La struttura del metodo che verrà richiamato dal client è:

```
    <WebMethod()> _
```

```
  Public Function GetSelection(ByVal search As String, ByVal field As String) As
                                                                          Indirizzi
...
   End Function
```

l'attributo <WebMethod()> indica infatti che il Web Service espone il metodo GetSelection , in fase di risposta l'oggetto di tipo Indirizzi che risulta dalla funzione sarà serializzato in XML secondo le indicazioni che sono state fornite negli attributi della classe Indirizzi che abbiamo visto prima.

Il risultato

Il resto del codice è molto simile a quanto visto negli esempi precedenti, il sorgente completo si presenta:

```
<%@ WebService Language="VB" Class="server" %>
Imports System
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.Odbc
Imports System.Collections.Generic
Imports System.Xml
Imports System.Xml.Serialization

<WebService(Namespace:="")> _
Public Class server
   Inherits System.Web.Services.WebService

   <WebMethod()> _
  Public Function GetSelection(ByVal search As String, ByVal field As String) As
Indirizzi
```

```
    Dim query As String = "SELECT CompanyName, ContactName, Address,
                                          City, Country FROM Customers "
    If Not String.IsNullOrEmpty(search) Then
        search = search.Replace("'", "''")
        If String.IsNullOrEmpty(field) Then field = "CompanyName"
        query &= String.Format(" WHERE {0} LIKE '{1}%'", field, search)
    End If
    Dim cnn As New OdbcConnection("DSN=NWIND;Uid=fs;Pwd=fs;")
    Dim cmd As OdbcCommand = cnn.CreateCommand()
    cmd.CommandType = CommandType.Text
    cmd.CommandText = query
    cnn.Open()
    Dim reader As OdbcDataReader =
                  cmd.ExecuteReader(CommandBehavior.CloseConnection)
    Dim result As New Indirizzi
    While (reader.Read())
        Dim cname As String = reader.GetString(0)
        Dim pname As String = reader.GetString(1)
       Dim address As String = reader.GetString(2) + " " + reader.GetString(3)
                                                        + reader.GetString(4)
        result.Add(cname, pname, address)
    End While
    Return result
End Function

<XmlRoot("root", Namespace:="")> _
Public Class Indirizzi
    Public Sub Add(ByVal company As String, ByVal contact As String, ByVal
                                                       address As String)
        Dim p As New Persona
        p.company = company
        p.contact = contact
        p.address = address
```

```
            Persone.Add(p)
        End Sub
        Private _Persone As List(Of Persona)
        <XmlElement("persona")> _
        Public Property Persone() As List(Of Persona)
            Get
                If _Persone Is Nothing Then
                    _Persone = New List(Of Persona)
                End If
                Return _Persone
            End Get
            Set(ByVal Value As List(Of Persona))
                _Persone = Value
            End Set
        End Property

        Public Class Persona
            <XmlAttribute()> Public company As String
            <XmlAttribute()> Public contact As String
            <XmlAttribute()> Public address As String
        End Class
    End Class

End Class
```

Se andiamo ad aprire il browser all'indirizzo della nostra pagina server.asmx troveremo un'interfaccia con un modulo che ci consente di testare direttamente il funzionamento del servizio, com'è possibile vedere in **figura 5**.
Se proviamo ad impostare i parametri e cliccare sul bottone "richiama" otterremo una risposta XML del tutto analoga a quella che abbiamo visto negli altri esempi, vedi **figura 6**.

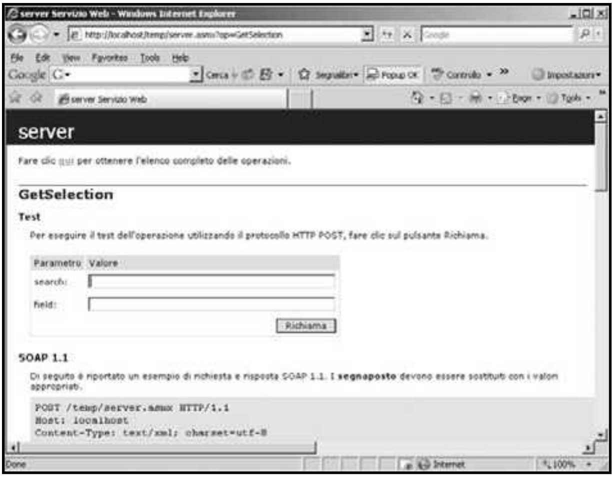


Figura 5: Interfaccia di test per il Web Service

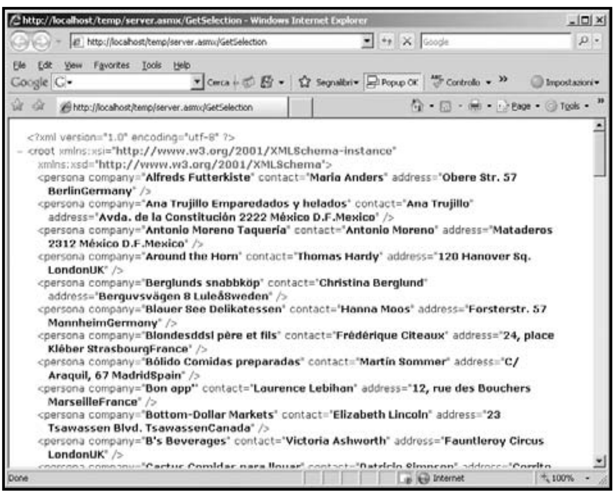


Figura 6: Output del Web Service

FINALMENTE AJAX

Tutto quanto visto fino adesso ci dovrebbe aver dato sufficiente bagaglio tecnico per affrontare una piccola applicazione in AJAX.
Come sempre partiamo da un problema specifico piuttosto che da una trattazione teorica.

5.1 IL PROBLEMA

Vogliamo fare un'applicazione che consenta la ricerca all'interno della tabella Customers del database Northwind, ci sarà una textbox dove l'utente inserirà i termini della ricerca, una select che consente di scegliere il campo in cui effettuare la ricerca e un bottone di invio.
Sul click sul bottone di invio dovrà partire una XMLHttpRequest verso una delle pagine lato server che abbiamo predisposto, dovremo poi interpretare i risultati e trasformarli in una tabella HTML per presentarli all'utente, il tutto senza ovviamente ricaricare la pagina.

5.2 SOLUZIONE

Prima di tutto dovremo creare una nuova pagina HTML. Per dimostrare come AJAX sia totalmente indipendente da tecnologie lato server, scegliamo di creare una pagina in HTML statico alla quale quindi daremo un'estensione .html o .htm.

Il layout HTML

Inseriamo nella pagina tutti gli elementi di partenza ovvero:

- una form contenente:
  - ✓ un textBox per inserire la stringa di ricerca
  - ✓ un select per selezionare il campo della tabella
  - ✓ un bottone submit
- un DIV vuoto destinato a contenere i risultati

Il layout risultante sarà:

```
<div class="title">Selezione tabella clienti</div>
```

```
<form onsubmit="sendRequest();return false">
<table class="layoutTable" cellpadding="2" cellspacing="0">
    <tr>
        <td>
        <div class="label">Parola iniziale</div>
        <input class="inputBox" type="text" id="search">
        </td>
        <td>
        <div class="label">Nel campo</div>
        <select id="field"></select>
        </td>
        <td valign="bottom">
        <input type="submit" class="submit" value="Invia">
        </td>
    </tr>
</table>
</form>
<div id="result"></div>
```

naturalmente avremo cura di attribuire degli id a quegli elementi che saranno coinvolti nel processo AJAX in modo da poterli facilmente referenziare nel codice javascript.
Il <form>, come vedete, non punta a nessuna pagina in particolare (manca infatti l'attributo action) ma serve solo a gestire l'evento onsubmit che esegue la funzione sendRequest (che definiremo poi) e quindi blocca l'invio effettivo con return false.
In realtà avremmo potuto anche assegnare direttamente la funzione sendRequest all'evento onclick di un bottone senza utilizzare per niente il <form> tuttavia, in questo caso, la funzione non sarebbe stata invocata alla pressione del tasto Return sulla casella di testo e questo sarebbe un comportamento che potrebbe disorientare molti utenti.
Un'altra particolarità è quella che la <select> che dovrebbe contenere la lista dei nomi dei campi è in realtà vuota.

Questo perché vogliamo riempirla dinamicamente, vediamo come.

La selectBox con i nomi dei campi

La query su cui ci basiamo è quella mostrata negli esempi lato server, e cioè:

```
SELECT CompanyName, ContactName, Address, City, Country FROM Customers
```

I campi su cui l'utente può basare la selezione sono:

- CompanyName
- ContactName
- Address
- City
- Country

Nell'XML risultante però i campi (attributi del nodo) sono solo tre : company (da CompanyName) , contact (da ContactName), e address (dalla concatenazione di Address, City e Country) come possiamo vedere da uno degli elementi dell'output:

```
<persona company="Alfreds Futterkiste" contact="Maria Anders"
                                address="Obere Str. 57 BerlinGermany"/>
```

Da ciò ne ricaviamo che:

- tutti i nomi dei campi devono apparire nella selectBox
- soltanto alcuni devono apparire nella tabella risultato

Quindi, in javascript andiamo a costruire un modello per definire i campi:

```
var fields = [
        {"value":"CompanyName","caption":"Nome azienda",
                                "attname":"company","header":true},
        {"value":"ContactName","caption":"Nome contatto",
```

```
                                        "attname":"contact","header":true},
    {"value":"Address","caption":"Indirizzo","attname":"address",
                                                        "header":true},
        {"value":"City","caption":"Città","header":false},
        {"value":"Country","caption":"Stato","header":false}
    ]
```

Cioè dichiariamo una variabile fields che contiene un Array di oggetti definiti inplace che hanno le seguenti proprietà:

- value – il nome del campo nella query SQL e valore nella <option> della <select>.
- caption – il nome convenzionale del campo che useremo per le intestazioni delle colonne e per il testo nella <option> della <select>.
- attname – il nome dell'attributo nel nodo <persona> che ci arriva dal server. Questo è presente solo nel caso in cui header sia true.
- header – indica se utilizzare l'oggetto per le intestazioni di colonna e per il recupero dati.

Certo che abbiamo complicato un po' le cose non è vero? Ma solo in apparenza, in realtà questi oggetti ci saranno molto utili:

- per riempire la selectBox
- per costruire le intestazioni colonna della tabella
- per recuperare i valori da inserire nelle celle della tabella

È vero, come obbietterà qualcuno, che nulla ci vieta di scrivere i valori della selectBox direttamente nei tag <option>, così:

```
<select id="field">
    <option value="CompanyName">Nome azienda</option>
    <option value="ContactName">Nome contatto</option>
    ...
</select>
```

Però se avessimo fatto in questo modo pensate all'eventualità in cui

dovessimo utilizzare un’altra query o cambiassero i nomi campi.
Disponendo invece dell’Array di oggetti fields possiamo riempire dinamicamente la selectBox, con questa funzione :

```
    function fillSelectFields(){
        //riferimento al <select>
        var oSel = document.getElementById("field");
        for(var i=0;i<fields.length;i++){
            var field = fields[i];
            //nuovo elemento <option>
            var oOpt = oSel.appendChild(
                                            document.createElement("option") );
            oOpt.value = field.value;
            oOpt.innerHTML= field.caption;
        }
    }
```

Invio della richiesta

L’invio della richiesta al server, con la libreria jsXML, è una cosa semplice:

```
function sendRequest(){
    var serverUrl = "server.php"; //o server.asp o server.aspx o server.asmx
    var searchValue = document.getElementById("search").value;
    var fieldValue = document.getElementById("field").value;
    //parametri
    var requestBody =  "search=" + encodeURIComponent(searchValue) +
                        "&field=" + encodeURIComponent(fieldValue);
    jsXML.sendAsyncPost(serverUrl,
                        requestBody,
                        formatResult,
                        error,
                        wait);
```

```
}
```

In pratica abbiamo indicato nella variabile requestBody i parametri selezionati (che derivano dall'inputBox e dalla selectBox) e abbiamo invocato una richiesta asincrona in Post con jsXML.sendAsyncPost.
A questo punto la logica si sposta sulle tre funzioni callback: formatResult, error, wait; le ultime due regolano rispettivamente il caso in cui si verifichi un errore o in cui ancora il flusso non sia terminato. Il loro codice sarà:

```
    function error (msg){
        alert(msg);
    }
    function wait(){
        document.getElementById("result").innerHTML= "Attendi..."
    }
```

ovvero: in caso d'errore mostra il messaggio e in caso di attesa scrive un avviso nel DIV di destinazione.
La parte di codice più articolata sarà ovviamente quella di gestione dei dati in caso di successo, ovvero la funzione formatResult:

```
function formatResult(strXML){
    var objDOM = jsXML.parseXMLDocument(strXML);
    var oResult = document.getElementById("result");
    oResult.innerHTML= ""; //cancellazione contenuto precedente
    var oTable = oResult.appendChild( document.createElement("table") );
    var oTr = oTable.insertRow(-1);

//costruisce le intestazioni di riga
    for(var i=0;i<fields.length;i++){
        var field = fields[i];
        if(field.header){
```

```
                        var oTh = oTr.appendChild( document.createElement("th") );
                        oTh.innerHTML= field.caption;
                }
        }
        //XPath per estrarre i nodi <persona>
        var persone = objDOM.selectNodes("//persona");
        for(var i=0;i<persone.length;i++){
                var persona = persone[i]
                var oTr = oTable.insertRow(-1);
                for(var j=0;j<fields.length;j++){
                        var field = fields[j];
                        if(field.header){
                                var oTd = oTr.insertCell(-1);
                                oTd.innerHTML= persona.getAttribute(field.attname);
                        }
                }
        }
}
```

Le operazioni in sequenza sono:

- trasformare l'input XML in forma di stringa in un documento DOM XML
- ottenere una referenza del DIV all'interno del quale andrà posizionata la tabella e cancellare l'eventuale contenuto
- creare un nuovo oggetto table e utilizzare gli oggetti dell'Array fields che hanno la proprietà header su true per costruire la riga dell'intestazione
- estrarre dal documento DOM XML i nodi <persona> e scorrerli in un ciclo for creando un riga della tabella ad ogni passaggio.
- All'interno del primo ciclo for innestare un secondo ciclo che torna a scorrere gli oggetti dell'Array fields , ne preleva la proprietà attname (nome dell'attributo) e la usa per estrarre il valore del corrispondente attributo del nodo <persona> e inserirlo in una nuova cella del-

la riga corrente.

Il risultato finale

Il codice completo della procedura AJAX sarà:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Esempio AJAX</title>
    <script language="javascript" type="text/javascript"
                                            src="jsXML.js"></script>
    <script language="javascript" type="text/javascript">
  var fields = [
        {"value":"CompanyName","caption":"Nome azienda",
                                  "attname":"company","header":true},
        {"value":"ContactName","caption":"Nome contatto",
                                   "attname":"contact","header":true},
    {"value":"Address","caption":"Indirizzo","attname":"address",
                                                       "header":true},
        {"value":"City","caption":"Città","header":false},
        {"value":"Country","caption":"Stato","header":false}
    ]
    function fillSelectFields(){
        var oSel = document.getElementById("field");
        for(var i=0;i<fields.length;i++){
            var field = fields[i];
            var oOpt = oSel.appendChild
                              ( document.createElement("option") );
            oOpt.value = field.value;
            oOpt.innerHTML= field.caption;
        }
    }
```

```
    function sendRequest(){
    var serverUrl = "server.php";
    var searchValue = document.getElementById("search").value;
    var fieldValue = document.getElementById("field").value;
    var requestBody =  "search=" + encodeURIComponent(searchValue) +
                          "&field=" + encodeURIComponent(fieldValue);
    jsXML.sendAsyncPost(serverUrl,
                          requestBody,
                          formatResult,
                          error,
                          wait);
}
    function formatResult(strXML){
        var objDOM = jsXML.parseXMLDocument(strXML);
        var oResult = document.getElementById("result");
        oResult.innerHTML= ""; //cancellazione contenuto precedente
        var oTable = oResult.appendChild(
                                  document.createElement("table") );
        var oTr = oTable.insertRow(-1);
        //costruisce le intestazioni di riga
        for(var i=0;i<fields.length;i++){
            var field = fields[i];
            if(field.header){
            var oTh = oTr.appendChild( document.createElement("th") );
                oTh.innerHTML= field.caption;
            }
        }
        //XPath per estrarre i nodi <persona>
        var persone = objDOM.selectNodes("//persona");
        for(var i=0;i<persone.length;i++){
            var persona = persone[i]
            var oTr = oTable.insertRow(-1);
            for(var j=0;j<fields.length;j++){
```

```
                    var field = fields[j];
                    if(field.header){
                         var oTd = oTr.insertCell(-1);
                         oTd.innerHTML= persona.getAttribute(field.attname);
                    }
               }
          }
     }
     function error (msg){
          alert(msg);
     }
     function wait(){
          document.getElementById("result").innerHTML= "Attendi..."
     }

     window.onload = function () {
          fillSelectFields();
     }
   </script>
     <link rel="STYLESHEET" type="text/css" href="style.css">
</head>
<body>
     <div class="title">Selezione tabella clienti</div>
     <form onsubmit="sendRequest();return false">
     <table class="layoutTable" cellpadding="2" cellspacing="0">
          <tr>
               <td>
               <div class="label">Parola iniziale</div>
               <input class="inputBox" type="text" id="search" value="">
               </td>
               <td>
               <div class="label">Nel campo</div>
               <select id="field"></select>
```

```
                  </td>
                  <td valign="bottom">
                  <input type="submit" class="submit" id="" value="Invia">
                  </td>
            </tr>
      </table>
      </form>
      <div id="result"></div>
</body>
</html>
```

Il risultato nel browser sarà quello visibile in figura 7, notate come ad ogni query la tabella venga popolata in background senza ricaricare la pagina.



Figura 7: AJAX in azione

5.3 PERSISTENZA DELLE INFORMAZIONI

L'applicazione già è funzionante, però un piccolo problema ancora c'è … se l'utente effettua per qualche motivo il reload della pagina (attraverso il tasto F5 oppure segue un link e poi ritorna nella pagina) tutto l'HTML generato dinamicamente scompare e si torna alla situazione

iniziale.
Intendiamoci, non sempre questo è un male, in questo caso ad esempio non è un vero e proprio problema quanto piuttosto un comportamento logico dell'applicazione.
Alcune volte però può esserlo, pensiamo a una tabella con paginazione: l'utente fa click su un prodotto che era a pagina 10 per vedere una scheda dettagliata, chiude la scheda e torna a pagina 1, questo decisamente è un comportamento non desiderabile.
In generale il problema della persistenza delle informazioni durante la navigazione è una delle questioni su cui ferve il dibattito tra i Pro e i Contro AJAX.
In realtà ci sono validi sistemi per risolverlo, tra questi:

- l'uso di frames nascoste
- l'uso dei cookies

frames nascoste

Forse il sistema migliore per garantire la persistenza delle informazioni durante la navigazione dell'applicazione è costituito dall'uso di frames nascoste, si tratta di incapsulare la pagina che ospita il codice AJAX in un frameset del tipo:

```
<html>
<head>
      <title>Applicazione AJAX</title>
</head>
<frameset rows="0,*" framespacing="0" frameborder="0">
      <frame src="hidden.html" name="hidden"                noresize/>
      <frame src="applicazioneAJAX.html" name="main"/>
</frameset>
</html>
```

come potete notare dall'attributo rows di <frameset> la prima frame sarà di altezza 0 e quindi invisibile.

Nel file della pagina invisibile, hidden.html, potremo inserire delle variabili a livello globale in uno script come :

```
<script language="javascript" type="text/javascript">
    var search = "sss";
</script>
```

dal codice della pagina visibile è possibile leggere e scrivere questa variabile con:

```
var value = top.hidden.search; //lettura
top.hidden.search = value; //scrittura
```

In questo modo la navigazione avverrà tutta all'interno del frame main e qualsiasi pagina che vi si trova sarà in grado di accedere alle variabili del frame hidden.
Il vantaggio di questa tecnica è che non è legata ai cookies (che potrebbero essere disabilitati dall'utente).
Lo svantaggio, per contro, è che non mette al riparo dal Refresh completo della pagina effettuato intenzionalmente o meno.

I cookies

Un altro sistema per garantire la persistenza è quello di usare i cookies ovvero le informazioni residenti in maniera più o meno temporanea nel computer dell'utente.
I cookies sono accessibili a javascript attraverso la proprietà document.cookie per gestire lettura è scrittura dei cookies si può usare del codice come:

```
var Cookies = new Object();
Cookies.get = function (name,defaultValue) {
    var nameEQ = name + "=";
    var ca = document.cookie.split(';');
```

```
    for(var i=0;i < ca.length;i++) {
        var c = ca[i];
        while (c.charAt(0)==' ') c = c.substring(1,c.length);
        if (c.indexOf(nameEQ) == 0) {
            var ckValue=c.substring(nameEQ.length,c.length);
            if(ckValue!='') return ckValue;
            }
    }
    return defaultValue;
}

Cookies.set = function (name,value) {
    var expires = "";
    var days = arguments[2];
    var path = (arguments[3])?arguments[3]:"/";
    if (days) {
        if(!isNaN(new Number(days))) {
            var date = new Date();
            date.setTime(date.getTime()+(days*24*60*60*1000));
            expires = '; expires="+date.toGMTString();
        }
    }
    var setValue = name+"="+value+expires+"; path=" + path;
    document.cookie = setValue;
}

Cookies.remove =  function (name){
    Cookies.set(name,"",-1);
}
```

in questo modo si hanno a disposizione le funzioni:

- Cookies.get – per leggere un valore dai cookies
- Cookies.set – per impostare un cookie

- Cookies.remove – per cancellare un cookie

Nella nostra applicazione AJAX dovremo prevedere una funzione che si occupa del salvataggio dello stato ad ogni richiesta ed una che si occupa di ripristinare lo stato precedente a ogni reload, come ad esempio:

```
function recordInput(){
	var inputs = ["search","field"];
	for(var i=0;i<inputs.length;i++){
		Cookies.set(inputs[i],document.getElementById(inputs[i]).value);
	}
}
function reloadInput() {
	var inputs = ["search","field"];
	for(var i=0;i<inputs.length;i++){
		document.getElementById(inputs[i]).value = Cookies.get(inputs[i],'');
	}
	if(Cookies.get("search",'')!='') sendRequest();
}
```

L’uso dei cookies per la persistenza ha come vantaggi :

- non richiede layout particolari come le frames
- può essere impostata anche una persistenza che va oltre la sessione
- resiste al refresh della pagina

L’aspetto negativo invece è solo uno, ma molto importante : se l’utente disabilita i cookies addio persistenza.
In generale comunque la persistenza non rappresenta poi quel problema che si vuole fare credere (anche perché spesso di usa AJAX come complemento a tecniche tradizionali); tuttavia in casi estremi, dove si vuole essere sicuri al 100%, nulla vieta di ricorrere a tutte e due le tecniche frame e cookies, in questo caso si avrà una doppia registrazione e un doppio controllo (cookies e frame)

5.4 NOSCRIPT

Un altro problema potrebbe essere il caso in cui l'utente abbia disabilitato gli script del browser, la soluzione in questi casi è avvertire l'utente del problema inserendo nella pagina un TAG <NOSCRIPT>, come ad esempio:

```
<NOSCRIPT>
<table width="100%" cellpadding="0" cellspacing="0">
    <tr>
        <td align="center">
        <table cellpadding="4" cellspacing="0">
            <tr>
                <td>
                <img src="gifs/warning.gif" border="0" align=
                                                "absmiddle" alt="">
                </td>
                <td style="font:bold 14px arial;color:red">
                Attenzione! il tuo browser ha disabilitato gli script.<br/>
                    L'applicazione funziona solo con gli script abilitati.
                </td>
            </tr>
        </table>
        </td>
    </tr>
</table>
</NOSCRIPT>
```

Il contenuto del TAG resta invisibile nel caso di scripts abilitati mentre avverte l'utente in caso contrario.

5.5 ALTRI MODI DI APPROCCIO AD AJAX – ASP.NET AJAX

Come abbiamo detto all'inizio di questo libro un po' tutte le "Major" del

software(come IBM, Sun e Microsoft), che talvolta oltre sviluppare linguaggi di programmazione producono anche ambienti di sviluppo, si stanno muovendo intorno al fenomeno AJAX cercando di integrare nei vari IDE soluzioni per gestire AJAX.
Emblematica in questo senso è la strada che ha scelto Microsoft con il nuovo Framework ASP.NET AJAX (http://ajax.asp.net).
In pratica questa soluzione aggiunge altri WebControls a quelli disponibili:

- che caricano nella pagina ASP.NET le librerie javascript necessarie
- forniscono un'interfaccia per la gestione di timer, pannello di caricamento ecc…

Oltre ai nuovi controlli abbiamo poi delle librerie lato client che "traducono" la risposta del server in oggetti javascript in maniera trasparente eliminando la necessità di usare XML.
Il lato server viene gestito con un Web Service come questo (l'esempio è ripreso dal sito sopra citato):

```
<%@ WebService Language="C#"
Class="Samples.AspNet.HelloWorldService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using Microsoft.Web.Script.Services;

namespace Samples.AspNet {

 [WebService(Namespace = "http://tempuri.org/")]
 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
 [ScriptService]
 public class HelloWorldService : System.Web.Services.WebService {

   [WebMethod]
```

```
  public string HelloWorld(String query)
  {
    string inputString = Server.HtmlEncode(query);
    if(!String.IsNullOrEmpty(inputString))
    {
      return String.Format("Hello, you queried for {0}. The "
        + "current time is {1}", inputString, DateTime.Now);
    }
    else
    {
      return "The query string was null or empty";
    }
  }
 }
}
```

Lato client invece il codice di gestione viene inserito in una pagina ASP.NET :

```
<%@ Page Language="C#" Title="ASP.NET AJAX Script Walkthrough" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml">
    <head id="Head1" runat="server">
   <title="ASP.NET AJAX Script Walkthrough" />
   <style type="text/css">
     body { font: 11pt Trebuchet MS;
       font-color: #000000;
       padding-top: 72px;
       text-align: center }

     .text { font: 8pt Trebuchet MS }
   </style>
```

```
  </head>
 <body>
 <form id="Form1" runat="server">
 <asp:ScriptManager runat="server" ID="scriptManager">
    <Services>
      <asp:ServiceReference path="./HelloWorldService.asmx" />
    </Services>
 </asp:ScriptManager>
   <div>
     Search for
     <input id="SearchKey" type="text" />
     <input id="SearchButton" type="button" value="Search"
       onclick="DoSearch()" />
  </div>
</form>
<hr style="width: 300px" />
<div>
<span id="Results"></span>
</div>
 <script type="text/javascript">
   function DoSearch()
   {
     var SrchElem = document.getElementById("SearchKey");
     Samples.AspNet.HelloWorldService.HelloWorld(SrchElem.value,
                                                    OnRequestComplete);
   }
   function OnRequestComplete(result)
   {
     var RsltElem = document.getElementById("Results");
     RsltElem.innerHTML = result;
   }
      </script>
</body>
```

```
</html>
```

Come possiamo vedere nella funzione javascript il metodo del Web Service è stato mappato, in maniera trasparente, su un oggetto javascript. Esempi di questo approccio naturalmente possono trovarsi in tutti i linguaggi. Beh … che dire?
Non c'è dubbio che queste soluzioni funzionino … a mio modesto parere tuttavia impararsi degli standard come XMLHttpRequest, DOM e XPath richiede più o meno lo stesso tempo che imparare queste API, con il vantaggio che, una volta che abbiamo capito il meccanismo, siamo liberi di implementare le soluzioni AJAX con il linguaggio lato server che preferiamo e addirittura possiamo costruirci delle API tutte nostre da riutilizzare nei progetti.

5.6 JSON IL "CUGINO" DI AJAX

Non possiamo parlare di AJAX senza fare nemmeno un cenno ad una tecnica molto simile e, per alcuni versi intercambiabile: JSON.
JSON è l'acronimo di JavaScript Object Notation ed è un formato di interscambio di dati basato su un particolare modo di scrivere gli oggetti. Un oggetto in javascript può essere definito in forma estesa come in:

```
var myObject = new Object();
myObject.width = 100;
myObject.height = 20;
myObject.name = "ABC";
```

oppure in forma contratta, come in:

```
var myObject = {"width":100,"height":20,"name":"ABC"}
La stessa cosa vale per gli Array che si possono definire come:
    var myArray = new Array();
    myArray[0]= "Mela";
```

```
    myArray[1]= "Pera";
    myArray[2]= "Banana";
```

o anche come :

```
    var myArray = ["Mela","Pera","Banana"];
```

Le notazioni contratte si possono anche combinare, per cui un oggetto che ha una proprietà di tipo Array, che con notazione estesa si sarebbe espresso:

```
var myObject = new Object();
myObject.width = 100;
myObject.height = 20;
myObject.frutti = new Array();
    myObject.frutti[0]= "Mela";
    myObject.frutti[1]= "Pera";
    myObject.frutti[2]= "Banana";
con notazione contratta può essere anche espresso come:
var myObject =
            {"width":100,"height":20,"frutti":["Mela","Pera","Banana"]}
```

Quindi possiamo avere infinite strutture e sottostrutture capaci di rappresentare una grande varietà di dati. A questo si aggiunga che javascript è dotato del metodo globale eval che consente di interpretare una qualsiasi stringa come codice javascript, pertanto scrivere:

```
var myObject =
            {"width":100,"height":20,"frutti":["Mela","Pera","Banana"]}
```

o scrivere:

```
eval('var myObject
```

```
={"width":100,"height":20,"frutti":["Mela","Pera","Banana"]}');
```

è la stessa cosa, tanto che potremmo poi invocare proprietà e metodi con la solita sintassi, come:

```
alert (myObject.width);
```

Se torniamo un attimo a quello che abbiamo visto AJAX vediamo come una volta che recuperati i dati dal server in formato XML questi dovevano poi essere letti con il DOM e XPath per essere trasformati in HTML.
Bene, l'idea di JSON è : perché il server invece di inviare i dati in XML non li manda direttamente in forma di stringa che Javascript possa interpretare come oggetto, come Array ecc…?
Ed infatti qui sta la differenza tra le due tecniche : l'uso di XML , in AJAX, e l'uso della notazione di javascript, in JSON.

AJAX vs JSON

JSON in confronto ad AJAX ha vantaggi e svantaggi.
I vantaggi di JSON sono:

- con JSON il flusso dei dati è più leggero perché la struttura a TAG di XML è più prolissa (anche se si può ridurre utilizzando, quando possibile gli attributi anziché gli elementi)
- con JSON si elimina la fase di parsing dei dati, basta solo XMLHttpRequest per il trasporto

AJAX invece ha dalla sua:

- i dati sono in un formato standard come XML, oggi li utilizziamo per un applicazione Web, domani li potremmo utilizzare come Web Service o da un'applicazione Desktop senza toccare il lato server
- il flusso XML in AJAX potrebbe anche essere bidirezionale, cioè non solo il client che riceve XML dal server, ma anche il server può ricevere XML dal client (e i linguaggi lato server sono certo più attrezzati per interpretare l'XML che una stringa di javascript)
- il test e debug sul server è più facile analizzando flussi XML rispetto

ad una stringa di javascript.

- il flusso XML oltre che analizzato e gestito con il DOM potrebbe essere anche direttamente trasformato con XSLT

In conclusione potremmo dire che sia AJAX che JSON sono due ottime tecniche, l'ideale sarebbe apprenderle entrambe per poi usare quella che si adatta di più alla situazione specifica.

AJAX ON THE ROAD

Dopo i primi approcci con la tecnologia AJAX, che abbiamo visto nel capitolo precedente è giunto il momento di vedere qualche esempio pratico di applicazione della tecnica a varie situazioni di "vita reale".
In primo luogo c'è da dire che con AJAX si possono sia sviluppare applicazioni complete che piccole applets (il termine naturalmente non ha niente a che vedere con le applet Java) di complemento ad applicazioni tradizionali.
Per applicazioni di complemento si intendono quelle "utility" che semplificano la vita all'utente del sito ma non sono essenziali per il suo utilizzo.
Come illustreremo meglio nel successivo capitolo, per i siti cosiddetti "pubblici" (cioè senza autenticazione utente) è preferibile utilizzare AJAX come supporto aggiuntivo alle procedure lato server.
Vediamo quindi un esempio di questa tecnica applicato al carrello di un sito Web di E-Commerce.

6.1 UN CARRELLO INTERATTIVO

In fase di ordine da parte del cliente normalmente si trova, nel carrello, una Form che consente di modificare le quantità di prodotto da ordinare.
Ora, è chiaro che in questa fase l'utente potrebbe inserire un valore corrispondente ad una quantità superiore a quella presente in magazzino.
La procedura, normalmente, deve controllare, in fase di acquisizione della richiesta (e quindi dopo il Post della Form) le disponibilità di magazzino e segnalare, di ritorno, l'eventuale errore.
Con AJAX ci proponiamo invece di segnalare l'eventuale superamento della quantità massima disponibile prima ancora che l'utente invii il Post al server.
Per l'esempio proposto, non volendoci legare a nessuna particolare tecnologia lato server abbiamo estratto un file XML con i dati presi dalla tabella Products del database di esempio fornito da Microsoft con SQL

Server, Northwind.
I nostri dati quindi sono su un file XML come questo:

```
<root>
<Products ProductID="1" ProductName="Chai" QuantityPerUnit="10 boxes
                    x 20 bags" UnitPrice="18" UnitsInStock="5" />
<Products ProductID="2" ProductName="Chang" QuantityPerUnit="24 - 12
                    oz bottles" UnitPrice="19" UnitsInStock="4" />
<Products ProductID="3" ProductName="Aniseed Syrup"
 QuantityPerUnit="12 - 550 ml bottles" UnitPrice="10" UnitsInStock="12"/>
<Products ProductID="4" ProductName="Chef Anton's Cajun Seasoning"
        QuantityPerUnit="48 - 6 oz jars" UnitPrice="22" UnitsInStock="3"/>
</root>
```

Nel file abbiamo una serie di elementi <Products> che rappresentano i prodotti, ognuno dei quali ha, come attributi:

- ProductID – ID del prodotto
- ProductName – Nome del prodotto
- QuantityPerUnit – Quantità per unità
- UnitPrice – Prezzo per unità
- UnitsInStock – Disponibilità di magazzino

Naturalmente, in un caso reale i dati e la risposta del server saranno generati dalla procedura (ASP.NET, PHP, JSP ecc…) di e-commerce, a noi interessa focalizzarci sul meccanismo lato client.
Quindi andremo a scrivere un file HTML con lo pseudo-codice di un ipotetico carrello con i prodotti corrispondenti ai dati contenuti nel file XML che abbiamo sul server.
La porzione di codice HTML in questione sarà pressappoco così:

```
<div class="title">Ordine</div>
<form>
<table  cellpadding="1" cellspacing="2">
```

```
<tr>
<th>Codice</th>
<th>Nome articolo</th>
<th>Unità per confezione</th>
<th>Prezzo per Unità</th>
<th colspan="2">Quantità</th>
</tr>
<tr>
<td>1</td>
<td>Chai</td>
<td>10 boxes x 20 bags</td>
<td align="right">18</td>
<td align="right"><input type="text" class="qta" id="prodotto1"
                                                    value="3"></td>
<td id="alert1"></td>
</tr>
<tr>
<td>2</td>
<td>Chang</td>
<td>24 - 12 oz bottles</td>
<td align="right">19</td>
<td align="right"><input type="text" class="qta" id="prodotto2"
                                                    value="2"></td>
<td id="alert2"></td>
</tr>
<tr>
<td>3</td>
<td>Aniseed Syrup</td>
<td>12 - 550 ml bottles</td>
<td align="right">10</td>
<td align="right"><input type="text" class="qta" id="prodotto3"
                                                    value="1"></td>
<td id="alert3"></td>
```

```
</tr>
<tr>
<td>4</td>
<td>Chef Anton's Cajun Seasoning</td>
<td>48 - 6 oz jars</td>
<td align="right">22</td>
<td align="right"><input type="text" class="qta" id="prodotto4"
                                                    value="5"></td>
<td id="alert4"></td>
</tr>
<tr>
<td colspan="6" align="right">
<input type="submit" class="submit" value="Invia">
</td>
</tr>
</table>
</form>
```

E il risultato nel browser sarà quello visibile in figura 8.
figura 8 – La form nel carrello (Immagini\8.psd)

Nella struttura HTML possiamo notare come :

- in ogni riga è presente una casella di testo per definire la quantità con ID corrispondente alla parola “prodotto” seguita dall’ID del prodotto corrispondente.
- a lato della casella di testo c’è una cella vuota con ID corrispondente alla parola “alert” seguita dall’ID del prodotto corrispondente.

Inseriamo quindi, nel codice della pagina HTML, il riferimento alla libreria di funzioni AJAX e XML di cui abbiamo parlato nel capitolo precedente, jsXML:

```
<script src="jsXML.js" type="text/javascript"></script>
```

ed impostiamo una funzione che controlla la disponibilità del prodotto esaminando il file XML di dati presente sul server (che avremo chiamato carrello.xml):

```
function controllaDisponibilita(inputBox){
        //controlla se il valore è un numero
        var value = new Number(inputBox.value);
        if (isNaN(value)) {
                alert("Il valore non è un numero valido");
                inputBox.value=0;
                return
        }
        var currentID = inputBox.id.replace("prodotto","");
        //Sub funzione di callback su ricevimento dati
        var dataArrive = function(strXML){
                //creazione del XMLDocument
                var doc = jsXML.parseXMLDocument(strXML);
                //individuazione del nodo prodotto
                var nodoProdotto =
        doc.selectSingleNode("//Products[@ProductID='" + currentID + "']");
                //recupero del valore
                var unitsInStock = nodoProdotto.getAttribute("UnitsInStock");
                //cella messaggio
                var alertCell = document.getElementById("alert" + currentID);
                //confronto valori
                if (unitsInStock < value) {
                        //se il valore è maggiore al max viene visualizzato
                                                        il messaggio
                        alertCell.innerHTML= '<span style="color:red">' +
                        'La quantità impostata è superiore al valore massimo di ' +
                                                        unitsInStock + '</span>';
                }
                else {
```

```
                    //se il valore è corretto viene cancellato un eventuale
                    //messaggio precedente
                    alertCell.innerHTML= '';
               }
          }
          jsXML.sendAsyncGet("carrello.xml","",dataArrive)
}
```

Questa funzione richiede come parametro l'elemento <input> HTML che contiene il valore della quantità ed effettua le seguenti operazioni:

- In primo luogo esamina se il valore della casella è un numero e, in caso contrario, avvisa l'utente.
- Poi ricava l'ID del prodotto corrente dall'id dell'elemento <input> che, come abbiamo detto, rispetta la convenzione di denominazione prodotto{n} dove {n} è l'id del prodotto.
- Quindi imposta una sotto-funzione, chiamata dataArrive, che verrà richiamata al ricevimento del file XML da parte del server e recupera il nodo del prodotto, legge il valore della quantità di merce in magazzino, lo confronta con quello impostato dall'utente e, eventualmente, inserisce un messaggio di errore nella cella adiacente.
- Da ultimo c'è la chiamata vera e propria della funzione contenuta nella libreria jsXML sendAsyncGet che si occupa di scaricare in modo asincrono i dati XML dal server e passarli alla sotto-funzione dataArrive in precedenza esaminata.

Non resta a questo punto che "agganciare" la funzione controllaDisponibilita ad un evento, scegliamo di collegarla all'evento onblur (quando l'utente esce dalla casella) di ogni casella che contiene le quantità, in questo modo:

```
<input onblur="controllaDisponibilita(this)" type="text" id="prodotto1"
                                                              value="3">
```

Notiamo che come parametro viene passato this ovvero il controllo cor-

rente.
In questo modo quando l'utente sceglie un valore superiore alla quantità massima verrà visualizzato il messaggio di cui in figura 9.
figura 9 – Il messaggio di errore in tempo reale (Immagini\9.psd)

Naturalmente questo è solo un esempio che non prende in considerazione il lato server, il file XML sul server infatti nel nostro caso è statico, ad esso va sostituita una pagina dinamica che colloquia con il database, vediamo come potrebbe essere in ASP.NET.

6.2 LATO SERVER

La pagina di controllo ha semplicemente ruolo di effettuare la query al database e restituire i risultati in formato XML, per far questo non c'è bisogno di creare una pagina ASPX, è sufficiente un Handler della richiesta in formato .ashx.
Il nostro Handler riceve come parametro l'id del prodotto da controllare, inserito nella Url, effettua una query a SQL SERVER utilizzando la clausola FOR XML e restituisce i risultati come XML.
Ecco tutto il codice necessario del nostro file checkCarrello.ashx :

```
<%@ WebHandler Language="VB" Class="checkCarrello" %>
Imports System
Imports System.Web
Imports System.Data
Imports System.Data.SqlClient

Public Class checkCarrello : Implements IHttpHandler

   Private ReadOnly Property IDProdotto() As Integer
      Get
         Return My.Request.QueryString("id")
      End Get
```

```
End Property

Public Function QueryXml() As String
    Dim conn As New SqlConnection("Data Source=MIOSERVER;Initial
                                Catalog=Northwind;Persist Security Info=True")
    Dim strSQL As String = "SELECT * FROM Products Where ProductID=@ID
                                                        FOR XML AUTO"
    Dim cmd As New SqlCommand(strSQL, conn)
    Dim param As SqlParameter = cmd.CreateParameter()
    param.ParameterName = "@ID"
    param.Value = IDProdotto
    cmd.Parameters.Add(param)
    Dim strResult As String = ""
    conn.Open()

    Dim reader As SqlDataReader =
                    cmd.ExecuteReader(CommandBehavior.CloseConnection)
    While reader.Read
        strResult &= reader.GetString(0)
    End While
    reader.Close()
    Return strResult
End Function

Public Sub ProcessRequest(ByVal context As HttpContext) Implements
                                            IHttpHandler.ProcessRequest
    context.Response.ContentType = "text/xml"
    With context.Response.Output
        .WriteLine("<root>")
        .Write(QueryXml)
        .WriteLine("</root>")
    End With
End Sub
```

```
    Public ReadOnly Property IsReusable() As Boolean Implements
                                            IHttpHandler.IsReusable
        Get
            Return False
        End Get
    End Property

End Class
```

Il testo XML di risposta prodotto da una Url come: checkcarrello.ashx?id=1 sarà:

```
<root>
<Products ProductID="1" ProductName="Chai" SupplierID="1"
                    CategoryID="1" QuantityPerUnit="10 boxes x 20 bags"
                UnitPrice="18.0000" UnitsInStock="39" UnitsOnOrder="0"
                                 ReorderLevel="10" Discontinued="0"/>
</root>
```

Naturalmente anche in questo caso si tratta di un esempio puramente esplicativo: nessuno mai si sognerebbe di mettere la stringa di connessione direttamente nel codice (c'è per questo il file Web.config) e sarebbe anche bene separare le query SQL o nella configurazione o in file distinti; facciamo così solo per brevità.
A questo punto l'unica modifica al codice AJAX sarà quella di cambiare la url di risposta del server, se prima era :

```
jsXML.sendAsyncGet("carrello.xml","",dataArrive)
```

e puntava al file statico, adesso sarà invece :

```
jsXML.sendAsyncGet("checkCarrello.ashx?id=" + currentID,"",dataArrive)
```

Dove currentID, lo ricordiamo, era il codice del prodotto incluso nell'id dell'elemento <input>

CONSIDERAZIONI FINALI E RIFERIMENTI

7.1 AJAX DOVE, QUANDO E PERCHÈ

Solitamente si usa affrontare l'argomento dei vantaggi e degli svantaggi di una tecnologia nel primo capitolo del libro.
Con AJAX abbiamo preferito spostare il tema in coda all'argomentazione perché su questa tecnica c'è ancora, crediamo, poca chiarezza e quindi sembrava più opportuno che prima venisse illustrata.

7.2 IN QUALI APPLICAZIONI USARE AJAX

AJAX è senza dubbio una tecnica affascinante, in grado di portare sul Web un'esperienza di usabilità e di interfaccia utente degna (e a volte superiore) delle applicazioni Desktop.
Tuttavia non è sempre detto che sia la soluzione migliore.
AJAX è infatti fortemente legata al client in quanto basata su Javascript, DOM HTML e XML, XPath e XMLHttpRequest, basta che manchi una di queste condizioni e l'applicazione non funzionerebbe.
È vero che la generalità dei browser oggi in circolazione è capace di gestire queste tecnologie, tuttavia, mentre lato server siamo in grado di controllare pienamente i requisiti (è chiaro che un'applicazione per .NET 2.0 non funzionerà se nella macchina è installato solo .NET 1.1 ecc…), lato client la situazione è molto più "aleatoria".
In generale, come regola "prudenziale" è consigliabile adottare AJAX:

- per applicazioni in aree non pubbliche del sito, come le sezioni di amministrazione, dove possiamo indirizzare gli utenti ver-

so l'uso di alcuni browser o l'abilitazione di alcune funzioni
- per le Intranet/Extranet, per le stesse ragioni di cui sopra
- per funzioni di "complemento", utilità che magari velocizzano il lavoro dell'utente ma non sono essenziali al funzionamento dell'applicazione

7.3 AJAX E I MOTORI DI RICERCA

Un'altra considerazione che porta a escludere l'uso di AJAX per l'homepage del proprio sito è data dall'impossibilità per i motori di ricerca di indicizzarne il contenuto, con AJAX (ma il discorso vale anche in generale per tutto l' HTML generato lato client e non scritto direttamente sulla pagina) il contenuto viene creato attraverso javascript, ma i robots dei motori di ricerca non eseguono gli script, per cui non riconoscerebbero il testo.
Quindi, se proprio non potete fare a meno di usare AJAX nell'homepage del vostro sito, non usatelo per mostrare informazioni chiave per la vostra attività.

7.4 I TOOLS

Di tools e librerie per AJAX ne esistono un po' per tutti i linguaggi di programmazione e gli ambienti di sviluppo, solo per citarne alcuni:
- **Google Web Toolkit** – http://code.google.com/webtoolkit - che traduce in AJAX applicazioni scritte in Java
- **Eclipse AJAX Toolkit Framework** – http://www.eclipse.org/atf - che integra tool di sviluppo AJAX nel popolare IDE
- **AJAX ASP.NET** – http://ajax.asp.net – già citato in precedenza, che rappresenta la proposta di Microsoft per l'integrazione di AJAX con ASP.NET e Visual Studio

Comunque si può tranquillamente affermare che, una volta apprese le basi concettuali della tecnica, l'unico tool che realmente è uti-

le è un buon editor HTML (come UltraEdit o Homesite) per il lato client e l'ambiente di sviluppo comunemente usato per il lato server.

7.5 RIFERIMENTI

Alcuni link utili sono :

http://www.w3schools.com – per AJAX e le altre tecnologie Web
http://www.topxml.com – con interessanti e completi tutorial sull'argomento
http://sourceforge.net/projects/sarissa - la popolare libreria per la gestione di XML con javascript
http://developer.mozilla.org/en/docs/AJAX:Getting_Started – un tutorial firmato Mozilla
http://msdn.microsoft.com/library - la famosa library di Microsoft che nella sezione Web offre numerosi e utili riferimenti per javascript e HTML
http://www.adaptivepath.com/publications/essays/archives/000385.php - l'articolo di Jesse James Garrett, colui che ha coniato il termine AJAX
http://www.xml.com – altro sito con interessanti articoli su XML e AJAX
http://www.w3.org – il sito del W3C che contiene tutti i riferimenti agli standard di cui abbiamo parlato

NOTE

i libri di

ioProgrammo

PROGRAMMARE CON AJAX

Autore: Francesco Smelzo

EDITORE
Edizioni Master S.p.A.
Sede di Milano:Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona ind. - 87036 Rende (CS)

Realizzazione grafica:
Cromatika Srl
C.da Lecco, zona ind. - 87036 Rende (CS)

Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinatore tecnico: Giancarlo Sicilia
Illustrazioni: Tonino Intieri
Impaginazione elettronica: Francesco Cospite

Servizio Clienti
☎ Tel. 02 831212 - Fax 02 83121206
@ e-mail: customercare@edmaster.it

Stampa: Grafica Editoriale Printing - Bologna

Finito di stampare nel mese di Gennaio 2007